Changes in Human Horizontal Angular VOR After the Spacelab SLS-1 Mission

by

Michael David Balkwill

BMath, University of Waterloo, 1989

Submitted to the Department of Aeronautics and Astronautics
in Partial Fulfillment of
the Requirements for the Degree of

MASTER OF SCIENCE
in
AERONAUTICS AND ASTRONAUTICS
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge, Massachusetts
February, 1992

Signature of Author _____
Department of Aeronautics and Astronautics
January 16, 1992

Certified by _____
Dr. Charles M. Oman
Thesis Supervisor

Accepted by _____
Professor Harold Y. Wachman
Chairman, Department Graduate Committee

Changes in Human Horizontal Angular VOR After the Spacelab SLS-1 Mission

by

M. David Balkwill

Submitted to the Department of Aeronautics and Astronautics
on January 16, 1992 in partial fulfillment of the
requirements for the Degree of Master of Science in
Aeronautics and Astronautics

ABSTRACT

The angular vestibulo-ocular reflex was investigated in four crew members of the Spacelab Life Sciences 1 (SLS-1) NASA shuttle mission. Testing was performed in four sessions prior to the flight, and in four sessions during the week after the landing. Subjects were seated upright and rotated in the dark at a constant angular velocity of 120 °/s for one minute, and then stopped, thereby stimulating primarily the horizontal semicircular canals. The velocity was shaped by an exponential (0.17 sec time constant) to produce a smooth stimulus. In one-half of the runs, the head remained upright after the stop; in the other half, the head was pitched forward 90° immediately after the stop. Eye position was recorded via EOG. Durations of the subjective sensation of rotation were recorded by an event button.

A software package was developed in C and MatLab to analyze the collected data. The analysis system was almost entirely automated. Slow phase eye velocity (SPV) was calculated using order statistic filters, dropouts in the SPV envelope were removed using a new statistical outlier technique, and the remaining SPV was fit to a five-parameter model (three free parameters) similar to a Raphan-Cohen velocity storage model. Button push data (subjective rotation durations), SPV profiles, and model parameters were each analyzed for differences within the pre-flight sessions, and between pre-flight and post-flight sessions. Button push and model parameter changes were investigated with two-sided and paired t-tests; SPV profile changes were investigated with simulated sum of t-squares distributions. The analysis methods developed for this project have potential application for clinical VOR analysis.

The dumping head movement produced a significant reduction in responses, as compared to the responses with the head maintained upright after the chair stop. All four subjects generated earlier post-rotatory button pushes and a reduced SPV. Model fits to one subject's data yielded significant reductions in all three free parameters: system gain (K), indirect pathway gain (g_0), and velocity storage time constant ($1/h_0$). The cupula time constant (T_c) and the neural adaptation time constant (T_a) were included in the model, but were fixed at 6 and 80 seconds respectively.

Responses were generally lower or unchanged immediately post-flight ("return" sessions) as compared to pre-flight, with a full or partial increase back toward pre-flight levels by the end of the post-flight week ("recovery" sessions). Button push times were significantly shorter in three subjects immediately post-flight, and were only slightly shorter in two subjects during the recovery sessions. Return SPV profiles were significantly reduced in one subject (no data for two subjects); recovery SPV profiles were slightly lower than pre-flight in all four subjects. Model fits (one subject) suggested that the decrease was due to a reduction in the indirect pathway gain, g_0.

One subject demonstrated a pre-flight directional asymmetry in the SPV profile, with the response to a CW rotational stimulus being reduced from that to a CCW stimulus. This asymmetry was not present during the return sessions, but was present in the recovery sessions. Model fits yielded an asymmetry in K of 24% pre-flight, -2% return, and 28% post-flight; g_0 and h_0 were not significantly altered.

Thesis Supervisor: Dr. Charles M. Oman
Title: Senior Research Engineer,
Department of Aeronautics and Astronautics

Acknowledgements

First thanks deservedly go to Jim for his technical expertise and assistance, and willingness ability to troubleshoot during crucial moments. Without him, it would have been impossible to perform the experiments upon which this work is based. It is this kind of help which is so essential, and yet always seems to be overlooked. And a fellow scotch drinker too! Sorry about the driving... cha cha cha!

Thanks to Sherry, for her role was equally important since she was the one in charge of the administrative side, making things running so smoothly that they were virtually unnoticeable. And yet she was more than this, acting as a friend and older sister to all of us. Also, she's one heck of a good cook! And real mean with a slot machine...

Thanks to Mark, Dan, and Brad as the "old hands" who taught us so much about both the science, the equipment, and the experiments. They made the confusing understandable.

Thanks to Beverly, Eleni, Kim, Tom, and Victoria for all of the secretarial help. And to Beverly also for the continuous and informative sports updates, particularly the soccer, as well as the scope of horror. Best of luck with the baby, Kim.

Thanks to Connie and Dan and the rest of the gang in purchasing for letting us get what we needed. Working in the real world has made me realize how wonderful and helpful you have all been.

Thanks to Stuart, Liz, Vanessa, Mel, Roger, and all the others at JSC and DFRF for all their help in the conduction of the experiments.

Thanks especially to Rhea, Drew, Millie, and Jim. You're dedication to the mission was exceptional, especially in-flight. The entire scientific community owes you a really b Thank You.

Thanks to Gail for all of her help, particularly Midwest Express! Congratulations on your promotion -- nobody deserves it more.

Thanks to Dan, Wally, Brad, and Chris for their help with the code. Your contributions shall forever live in the way of comments :-) And to Ted for his most recent help with the manual and some of the code.

Thanks to Michele and Michelle and Chris for the help they gave me as UROPs. Things were often frustrating, but your assistance was indeed appreciated.

Thanks must also be given to Dr. Natapoff for his extensive and precise suggestions on both the statistical and grammatical aspects of my thesis. Without our detailed sessions, it would not be the absolutely crystal-clear work of art which it is.

Thanks to Vic, Craig, Andy, Nick and the rest of the guys on the soccer and volleyball teams. We kicked some butt, and burned off some frustrations by doing it. Knock 'em dead next year -- Go Eagles!

Thanks to Ron, Joshua, Christine, and the rest of the MIT Coalition Against Apartheid for keeping myself and the community in general informed of the needs of the world. It was even fun getting arrested together. Power to the people! Amandla Ngawetu!! Special thanks to Corrie, who is one of the most genuine and caring people that it has ever been my pleasure to meet, and an incredible dynamo who never hesitates to speak out (and act too!) on behalf of the downtrodden. You have taught me so much about giving a damn. The more people there are like you, the better chance the world has of surviving.

Thanks to Gail, Tom, Cheryl, Brad, and Sen for the old times, and congratulations on the greener pastures. Most recently, to Keoki, Jock, and Glenn -- it was lots of fun diving face first in the sand. Nick and Dava, hurry up and join them! Best wishes to the new hamsters: Valerie, Ted, Karla, Juan, and Scott.

Thanks to TJ's BVDS for the survival juice! It made the days and nights and nights and the next days and the next nights conscious. Sorry to stick you guys without a nifty group name any more. You need another D but I don't think she's in town enough to qualify! Best of luck to you all.

Thanks to Ginny too for making the lab meetings enjoyable. I'll miss the yum-yums.

Thanks to Rick and Amy for the beers and the cheers. Good true friends are very hard to come by, and it's very good to have some that you can really talk to. Let's do it again real soon, and real often. By the way, Amy, you still owe me a flight.

Thanks to Neil and Geddy and Alex for the all-night motivation music -- the best band ever. And to the others too numerous to mention. A special dedication to SRV who was lost, but shall never ever be forgotten. It seems the best came after you were no more.

Thanks especially to Andrew who has been the best friend anyone could ever have, and my best sounding board. The darts, the snooker, the beer, the etc. (I promise not to mention control theory again). The world looks so much better from 5000 feet, doesn't it? Someday we may be able to split the "driving" properly -- I wonder if they have a Cessna Aerobie yet? The control actuators might be a little difficult to design though...

Finally and foremost, extra incredible special thanks go to Barbara and Wendy for making my life full of happiness and meaning. You mean so much to me, there are no words that even begin to describe it. And yes I checked the whole thesaurus! You brought me up when I was down, and dragged me away on vacations when I needed them (and I complained so loudly too :-). The camping trips were so enjoyable, and Winchester will never be forgotten. Everything was always so natural and so right. You have also brought me into some parts of the stores I never thought I would set foot in. I could go on and on, but we'd rather do it in person. Always and forever, and forever and always, the three of us -- maybe more? And your little dog too!!!

So long, and thanks...
Wonko II

A final note to the financial gods who made this work possible. This research was funded by NASA Contract NAS9-15343, "Spacelab SLS-1/E072", and USRA 905-62, "Spacelab IML-1 MVI Experiment".

Table of Contents

List of Figures

List of Tables

1. Introduction

Human space exploration is a real part of today's world as mankind tries to expand the frontiers of knowledge. Current astronauts use the space shuttle to perform experiments and launch satellites. Long term plans include the establishment of a permanently manned space station, with the possibility of building settlements on the moon and Mars. All of these environments differ substantially from the terrestrial environment to which we have all become accustomed. Therefore, a good understanding is needed as to how the human body will react to these new conditions.

It has long been known that most astronauts experience physical discomfort, ranging from sweating and stomach awareness to violent vomiting episodes, upon initial exposure to space (Davis et al, 1988). These symptoms vary in severity and duration from one astronaut to another, but they usually decrease gradually and disappear within three to five days. This is known as space motion sickness (SMS).

Similar symptoms have been evoked in parabolic flight (Lackner and Graybiel, 1984), centrifuge studies (Guedry and Benson, 1978), and high-performance aircraft, the common element being the presence of an unusual gravitational force. On earth, the body is used to a roughly constant gravito-inertial force (referred to as G) caused by an acceleration of 9.8 m/s^2. The various sensory systems (visual, vestibular, proprioceptive) which yield position and orientation information are accustomed to this 1-G bias. Current theory is that an unusual gravito-inertial force causes unusual and conflicting signals from the sensory system (Oman et al, 1986). The central nervous system (CNS) is unable to convert these signals into a picture of the body orientation, and it is thought that this causes the discomfort and illusions. This is known as

sensory conflict theory. As the altered gravitational field persists, the CNS gradually learns how to reinterpret the information.

In an effort to better understand this process, a series of experiments were designed for the Spacelab Life Sciences 1 (SLS-1) shuttle mission in June, 1991. One particular experiment — the subject of this work — involves the use of a rotating chair to study the effects of yaw stimulation to the human body, focusing on the behaviour of the vestibular system. Experiments were performed in 1-G before flight, in weightlessness, and in 1-G immediately after adaptation to weightlessness. However, only those performed before and after the shuttle flight will be discussed here, since the in-flight data was obtained with different equipment and methods from the ground-based studies. Similar experiments have been performed previously on the SL-1 and D-1 missions and have indeed shown some changes in the way that the CNS interprets the various sensory information after exposure to weightlessness (Kulbaski, 1986; Oman and Weigl, 1989). The SLS-1 experiments include improvements in the design of the protocol, the acquisition of the data, and the subsequent analysis. The data will be compared against mathematical models to quantitatively determine the exact nature of the changes which occur in the sensory and central nervous systems as the body adapts to a new G-level.

1.1 Thesis Organization

Chapter 2 reviews the relevant physiology of the sensory systems and previous research into its functioning. Various methods of analyzing the data will also be discussed.

Chapter 3 outlines the experimental protocol and the methods used to record the data.

Chapter 4 describes the data analysis algorithms which were developed for this project.

Chapter 5 presents the results from the experiments described in Chapter 3.

Chapter 6 presents discussions and conclusions as to the nature of the changes which occurred, along with some recommendations for future work in both the experimental and the data analysis realms.

2. Background

Body position and orientation is determined by the central nervous system (CNS) from a number of different sensory input systems. Much research has been performed in an effort to understand this process, ranging from invasive studies at the neural level to exterior observations of whole-body locomotion.

2.1 Physiology of the Vestibular System

Angular and linear motion are measured by the semicircular canals and the otoliths respectively (Wilson and Melvill Jones, 1979). These two sensory organs are the primary components of the vestibular system. A set of three semicircular canals and two otoliths comprises a labyrinth (see Figure 2.1). Each set of canals is roughly orthogonal and is tilted approximately 20° upwards from the horizontal plane of the head. The canal which is closest to the horizontal plane will be referred to as the horizontal semicircular canal. The otoliths are generally approximated to be flat plates, one oriented horizontally (utricular otolith) and the other vertically (saccular otolith). One such labyrinth is contained within the inner ear on each side of the head; if one labyrinth is being discussed, the "complementary" labyrinth refers to the one in the opposite ear.

The semicircular canals sense angular motion of the head. Each canal contains a ring of endolymph fluid and a cupula (see Figure 2.2a). When the head is rotated in the plane of the canal, the moment of inertia of the fluid causes it to lag behind the rotation, deflecting the cupula in the opposite direction to the motion. There is also a restoring force due to the physical connection between the cupula and the ampulla which attempts

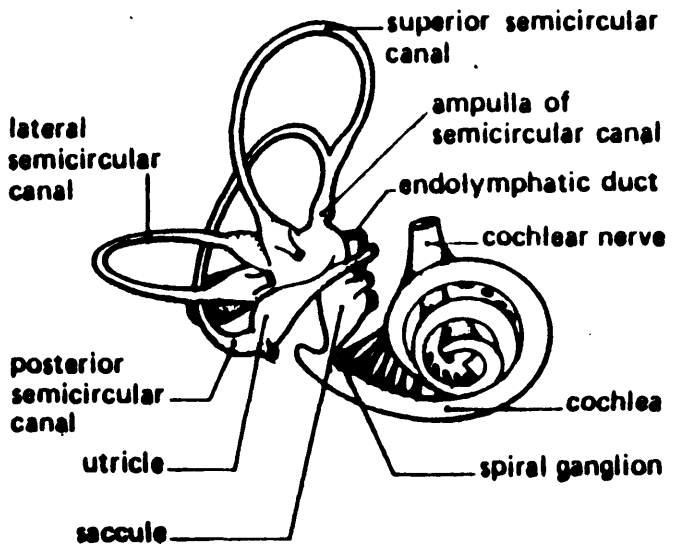


Figure 2.1. Membranous labyrinth of the right ear. [From Laurence Urdang, 1982]

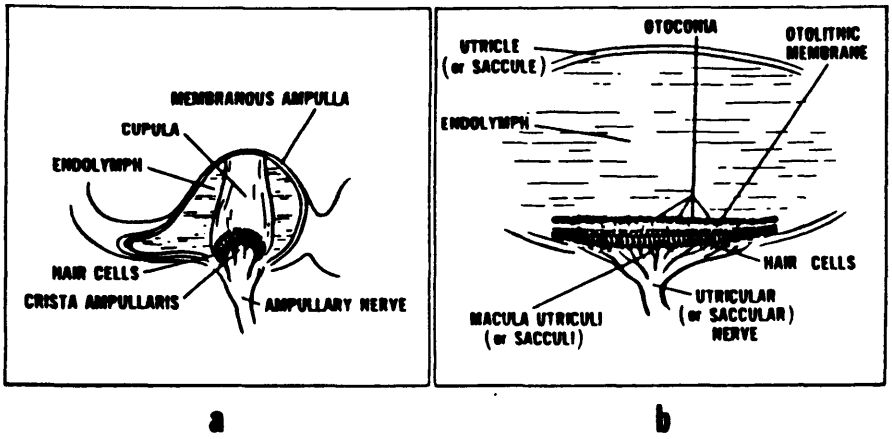


Figure 2.2. The vestibular end-organs. a. The ampulla of a semicircular duct, containing the crista ampullaris and cupula. b. A representative otolith organ, with its macula and otolithic membrane. [From Gillingham and Wolfe, 1986]

to return the cupula to the resting (zero) position. The resultant motion is that of a damped pendulum. The angular deflection of the cupula is initially proportional to the angular velocity of the head; then, if that velocity is maintained, the cupula deflection decays exponentially to the zero position. As the cupula deflects, the bending of the hair cell cilia causes a proportional direction-dependent change in the firing rate in the associated neurons from the normal resting firing rate. Rotation in a given direction is excitatory (increased firing rate) in one canal and inhibitory (decreased rate) in the complementary canal. If the velocity is too large, the firing rate along one ampullary nerve may under- or over-saturate.

The otolith organs (see Figure 2.2b) sense the linear acceleration of the head and the direction of gravity. The sensory receptor sites, the maculae, contain a layer of calcium carbonate crystals (otoconia) embedded in the "otolithic membrane". If the head is linearly accelerated in the plane of the macula, the inertia of the otoconial membrane will cause it to lag behind the motion, bending the underlying hair cell cilia in the opposite direction. If the head is tilted in either roll or pitch, this will cause the otoconial membrane to shear as if the head had been linearly accelerated; this is because the direction of gravity has changed relative to head-fixed axes, and gravitational forces are resolved similarly to any other linear acceleration. There is no transduction of a force perpendicular to the plane of the macula. In the absence of linear acceleration of the head, the pair of otoliths detect the magnitude and direction of gravity. If gravity is absent, as is encountered in space, the otoliths measure only linear acceleration. If both are absent, the otoconia are free-floating within the bounds of their physical connections.

2.2 Vestibulo-ocular Reflex

The vestibular system is the primary sensor of head motion within an inertial reference frame. The visual system also senses head motion, but with respect to the surrounding view. Proprioceptive, tactile, and auditory cues also provide information about the motion of the head and body, both with and within its surroundings. The CNS then takes all of these sensory inputs and estimates the position and orientation of the body.

When the head is rotated in the light, the eyes must be rotated in the opposite direction to stabilize the visual image on the retina. The retinal slip hypothesis states that the rate at which the image moves on the retina acts as an error signal, driving the compensatory eye movements to minimize this error signal. Information from the vestibular and proprioceptive systems can be used to estimate the velocity at which the head is being moved, with the retinal slip fine-tuning this estimate. Slow small movements can be easily tracked and compensated. However, there are physiological limitations on the degree to which the eyes may rotated. If the motion exceeds these bounds, the eyes will generally quickly jump in the direction of the motion to a new position, and will resume tracking; visual information is suppressed during these jumps. A repeated sequence of these reflexive slow tracking motions (slow phases) followed by quick jumps (fast phases) is known as optokinetic nystagmus (Henn et al, 1980). The velocity of the eyes during the slow phases, known as the slow phase velocity (SPV), is approximately equal to the velocity of the visual scene relative to the head.

This phenomenon also occurs when the head is moved in the dark, despite the fact that there is no retinal slip error signal. When there is no visual input, the CNS bases its estimate of the head velocity almost entirely on vestibular information, and drives the

oculomotor muscles according to this estimate. This is known as the vestibulo-ocular reflex (VOR). Prolonged motion of the head in one direction in the dark will cause vestibular nystagmus in which the SPV is proportional to this estimate.

The VOR is very useful to researchers, since it is much more convenient to measure the position of the eyes through electro-oculography (EOG) or video methods than it is to make direct neural recordings. Differentiating the eye position yields the eye velocity, from which the SPV can be extracted. Theoretically, this should indicate the internal estimate of body motion.

It is interesting to note that the VOR can be affected by the mental image which the subject has about the real world. In particular, if a point is imagined to be moving with the subject in the dark, and vision is fixated on that imagined point, the nystagmus will be suppressed to some extent. Therefore, it is important to give careful and uniform gaze instructions to human subjects when using the VOR in experimental tests.

2.3 Velocity Storage

Nystagmus recordings have shown that the internal estimate of angular velocity, as reflected through angular VOR, involves more than a direct pathway from the semicircular canals. Studies have been performed in which monkeys were rotated in the dark for a period of time and then stopped abruptly, resulting in post-rotational nystagmus. Direct recordings from the primary afferent neurons of the semicircular canals returned to resting firing rates significantly before the nystagmus ceased. Other studies involved the rotation of a visual scene about a stationary subject (Cohen et al, 1977); when the lights were extinguished, nystagmus persisted for several seconds

despite the fact that all sensory organs should indicate either a complete lack of motion or no information.

As a result of these observations, the existence of a velocity storage element has been hypothesized. This element extends the duration of the estimated velocity, perhaps in an effort to compensate for the cupula restoring force in the semicircular canals. The exact physiological nature and location of the velocity storage is unknown. However, direct neural recordings of the vestibular nucleus have shown the presence of units corresponding to both the afferent signals from the peripheral sensory organs and the signals as modified by the velocity storage. Therefore, it is likely that the velocity storage element is present in some combination of the vestibular nucleus, cerebellum, and connecting feedback loops. The effects of the velocity storage on optokinetic nystagmus and after nystagmus are extinguished in labyrinthectomized monkeys, demonstrating that the velocity storage is controlled by the vestibular system (Cohen et al, 1973).

2.4 Horizontal VOR Response

The response of the horizontal semicircular canals and velocity storage can be isolated from the other senses by rotating a subject in the dark, with the head erect, about the longitudinal axis. This is commonly done by seating the subject in a rotating chair. Such a stimulus would cause the utricular otoconial membranes to shear away from the body longitudinal axis, unlike the shearing due to a change in linear acceleration. Careful design to mask auditory and proprioceptive cues will usually effectively eliminate all sensory inputs except for the semicircular canals.

Under these conditions, a step in angular velocity from zero to a constant level Ω will generate horizontal per-rotatory nystagmus whose slow phase velocity reaches an initial peak magnitude which is proportional to Ω. This constant of proportionality varies amongst and within species. For human subjects, which is our main concern, this factor usually ranges from 0.5 to 0.8. The cupula position then returns exponentially to its resting position, as inferred from primary afferent neurons in various species. The time constant of this decay also varies amongst species. Analyses of results from a number of experiments, combined with estimates based upon the flow dynamics within the canals, have resulted in an estimate of this time constant on the order of 6 seconds.

After a period of approximately 40 seconds, the cupula will have returned to its rest position. If the rotation is subsequently stopped, the stimulus to the semicircular canal is equivalent to an equal and opposite change in velocity. This results in post-rotatory nystagmus which is a mirror image of the per-rotatory nystagmus.

Figure 2.3 shows a typical SPV response (solid line) of the Raphan-Cohen model (see Section 2.6) for this class of stimulus (dashed line). The velocity storage element lengthens the duration of the SPV in both the per- and post-rotatory segments, altering the response from a simple exponential. There is also a neural adaptation with a time constant on the order of 80 seconds which can cause the SPV to reverse direction during the rotation (Young and Oman, 1969).

A common modification to this stimulus is to pitch the head forward subsequent to the end of the rotation, evoking sensory conflict. The horizontal semicircular canals indicate rotation of the head about an off-vertical axis due to the stop. If this were truly the case, the direction of gravity would be changing with respect to head-fixed axes. However, the otoliths indicate a simple tilt, with a constant gravitational force. This

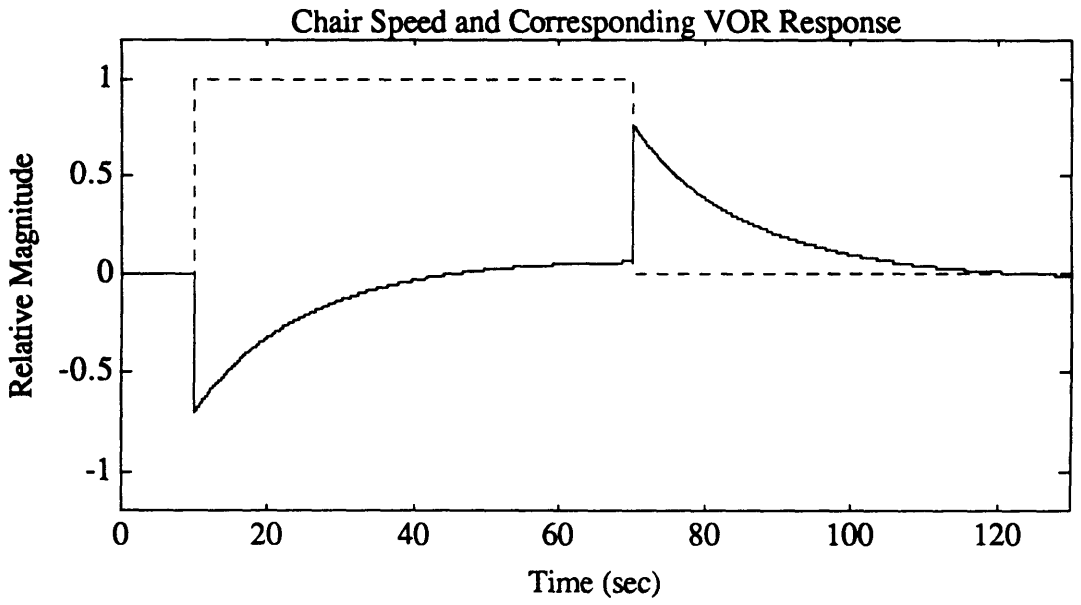


Figure 2.3. Theoretical slow phase velocity response to a step in velocity. Dashed line is the angular velocity of the stimulus, solid line is magnitude of SPV relative to stimulus velocity, as calculated by Raphan-Cohen model (Raphan et al, 1977).

conflict persists until the cupula returns to its rest position. Such a head movement has the effect of drastically shortening the time course of the post-rotatory nystagmus. It is hypothesized that the sensory conflict is responsible for this shortening, although the exact mechanism by which this happens is unknown. The most popular theory is that the velocity storage component of SPV is "dumped" out (partially suppressed), thereby producing a profile similar to the six second time constant of the canals alone. For this reason, such a head movement is referred to as a "dumping" manoeuvre. Alternatively, it could be that the axis of rotation of the velocity storage vector becomes aligned somewhere between the original axis of rotation (earth-vertical) and the new head axis. This would decrease the magnitude of the projection of the velocity storage vector on the head-fixed axis, thereby reducing the horizontal SPV. The remaining component of the velocity storage would manifest itself as SPV about the original earth-vertical axis, in this case producing torsional nystagmus.

2.5 Exposure to Altered Gravity

In the last ten years, research has begun in earnest into the effects on SPV of different gravitoinertial force (GIF) levels. The above experiment was performed in parabolic flight (Dizio et al, 1987; Dizio and Lackner, 1988) to investigate the SPV response in 0 G, 1 G, and 1.8 G. By fitting a single exponential to the post-rotational data, they discovered that the time constant of decay was shorter in both 0 G (significant) and 1.8 G (trend only) than it was in 1 G. With a dumping head movement, this time constant was significantly reduced in 1 G and 1.8 G, with no significant change in 0 G. It would appear that the presence of an abnormal GIF dumps the SPV in the same way as a pitching head movement, and that weightlessness was sufficiently abnormal to dump all of the stored velocity without requiring a head movement. It should be noted that the peak SPV was the same at all GIF levels, but was significantly less than the peak value in 1 G on the ground; hence, there is some difference to be attributed to the nature of parabolic flight itself. This may simply be due to decreased alertness or feelings of nausea, both of which are known to cause such a change.

The effects of extended weightlessness on horizontal VOR response has been investigated as part of the Spacelab SL-1 (1983) and D-1 (1985) missions. Experiments were performed on the ground before and after both missions, with additional in-flight tests on the D-1 mission only. On SL-1 (Kulbaski, 1986; Oman and Kulbaski, 1988), four shuttle astronauts were tested on five separate days before the flight, and on the first, second, and fourth days after the landing. They were subjected to two types of vestibular stimuli, head-up and dumping runs, consisting of a one minute spin in a rotating chair followed by a sudden stop. In the head-up runs, the head was held upright for one minute during the spin (per-rotatory) and for one minute after the stop (post-rotatory). In the dumping runs, the head was pitched forward 90°

five seconds after the stop and brought back upright after a further five seconds. Tests were performed with both clockwise and counter-clockwise rotations. Eye position was recorded through electro-oculography, and the SPV envelopes for the post-rotatory portions were calculated. Data was averaged across all subjects for the five pre-flight sessions to obtain a pre-flight group mean, and across the first two post-flight sessions to obtain a post-flight group mean. A first order model (single exponential) was fit to the head-up SPV data between 1 and 20 seconds, and to the dumping SPV data between 5 and 10 seconds. The head-up time constant decreased significantly after exposure to weightlessness (11.7 seconds pre-flight to 9.3 seconds post-flight), with no change in the dumping time constant (3.2 seconds to 3.4 seconds) or the gain (0.59 to 0.60). On the last post-flight session, the SPV responses had returned to their pre-flight level. A χ^2 test was used to compare SPV profiles, and demonstrated significant differences between the pre- and post-flight SPV from 6 to 20 seconds after stop (see Figure 2.4), and between the head-up and dumping SPV from 5 to 10 seconds. The χ^2 parameter is a sum of squares of normally distributed parameters, Z^2 (Breiman, 1973).

On D-1 (Oman and Weigl, 1989), five crewmembers were tested on four pre-flight and four post-flight days. The experimental methods were identical to those of the SL-1 mission. Only the post-rotatory head-up data was analyzed, but the results were consistent with those of SL-1. The time constant of decay was significantly reduced from pre-flight levels on the first post-flight day, while the gain was unchanged. A χ^2 test demonstrated a significant difference between pre- and post-flight SPV (see Figure 2.5). Two of the five subjects were also found to have a directional asymmetry; the peak response to rotation in one direction was similar to that of the other three subjects, but the peak response in the opposite direction was significantly reduced.

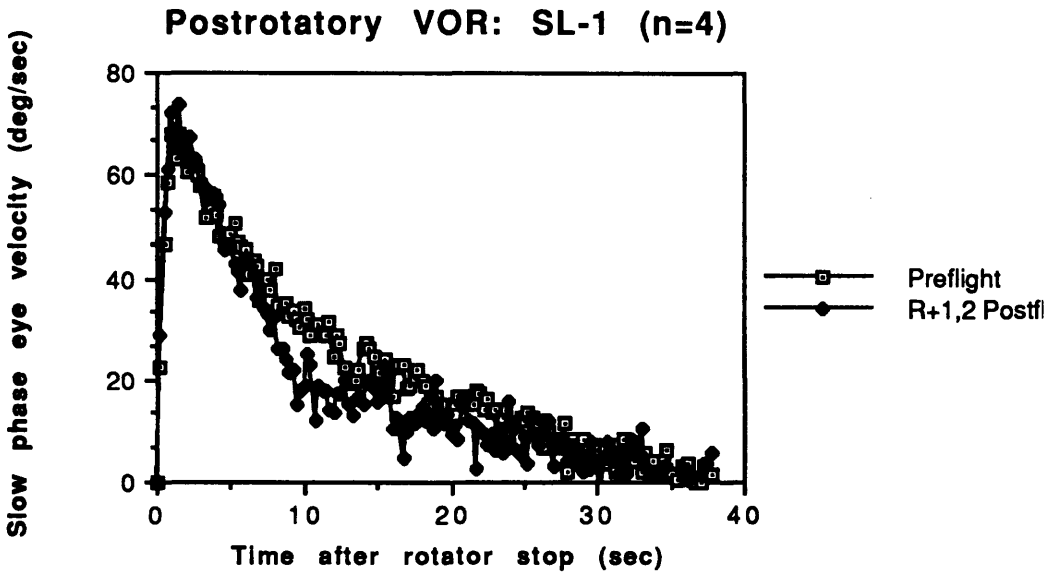


Figure 2.4. Group mean head-up post-rotatory SPV over four subjects, SL-1 pre-flight and post-flight. [From Oman and Weigl, 1989]

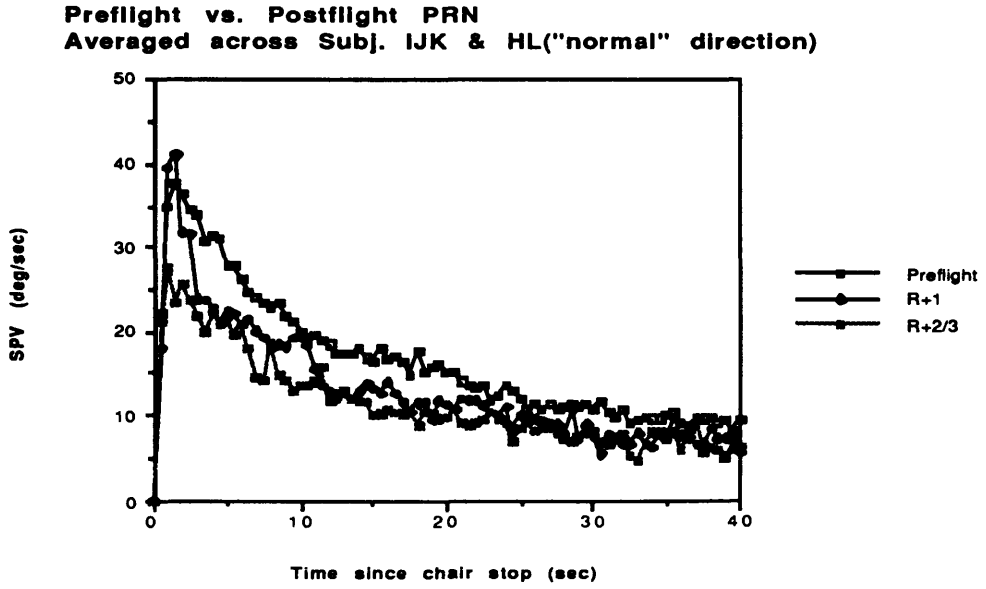


Figure 2.5. Group mean head-up post-rotatory SPV over five subjects, D-1 pre-flight and post-flight. [From Oman and Weigl, 1989]

2.6 Formal Models

The various aspects of the horizontal angular VOR response have been described above; they are formally incorporated in the model of Raphan et al (see Figure 2.6). This model is simplified if the rotation is performed in the dark, and it is assumed that the subject is not fixating on an imagined target. The velocity storage component is indicated by the indirect pathway containing a leaky integrator. If the velocity storage is assumed to be symmetric (i.e. $g0_L = g0_R$ and $h0_L = h0_R$), then the model can be expressed easily in Laplace transfer function form.

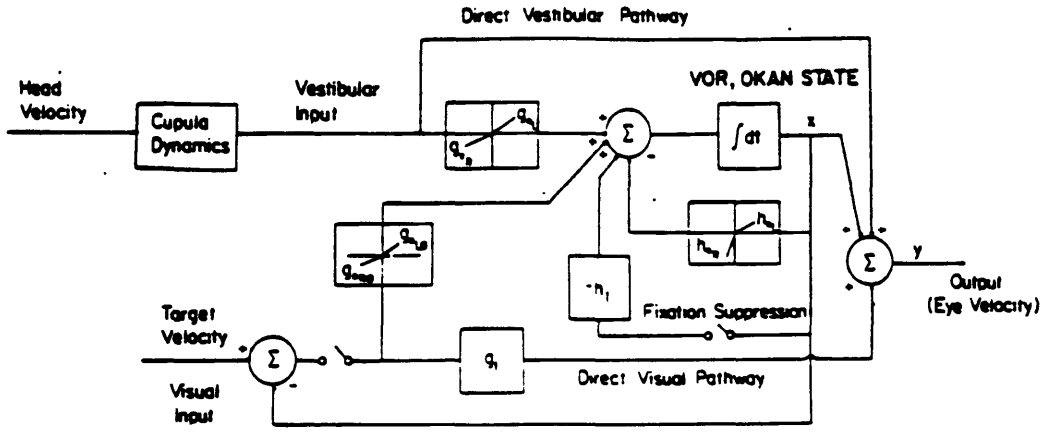


Figure 2.6. Raphan-Cohen model of OKN, OKAN, vestibular nystagmus, and visual-vestibular interaction. [From Raphan et al, 1977]

The cupula dynamics in the Raphan and Cohen model assume a simple exponential decay, with a system gain K and a time constant T_c characteristic of the cupula (about 6 seconds). For our purposes, the adaptation effects are included as a cascade term with a single time constant T_a. This model is shown in Figure 2.7. g_0 and h_0 are the indirect pathway gain and the leak rate, due to the velocity storage integrator. Notice that this model is open-loop due to rotation in the dark since there is no retinal slip feedback.

If the head undergoes an angular velocity in excess of about 60 °/s, the firing rate in the inhibitory vestibular neuron will vanish. Thus, the signal from the canals for large amplitude angular head movements will be based upon information from only one side. If a labyrinth suffers neurological damage, the magnitude of the SPV will be reduced in this direction (Baloh et al, 1977), producing a directional asymmetry in the response.

There are other possible causes for such an asymmetry, including oculomotor irregularities such as esophoria and exophoria, which could affect the SPV at lower angular velocities as well. It is difficult to model the myriad of possible factors; so, for the angular VOR model used in this thesis, the asymmetry is modelled simply as a difference in gain between the two directions. This non-linearity is reflected by the asymmetry parameter, A, in Figure 2.8 immediately after the cupula dynamics, and hence is included in both direct and indirect pathways.

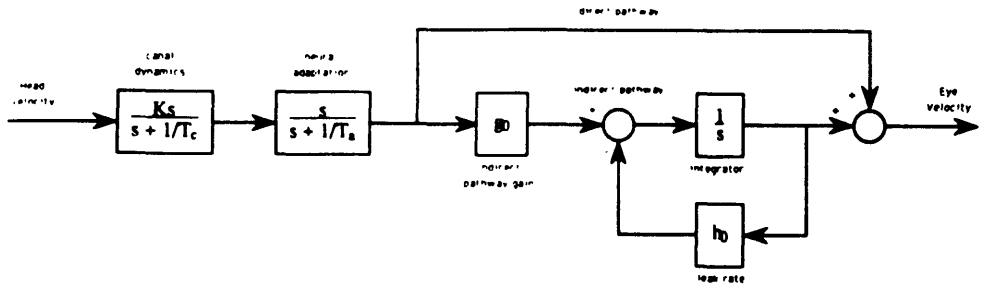


Figure 2.7. Symmetric Laplace transfer function model of vestibular nystagmus, based upon the Raphan-Cohen model, with no visual input.

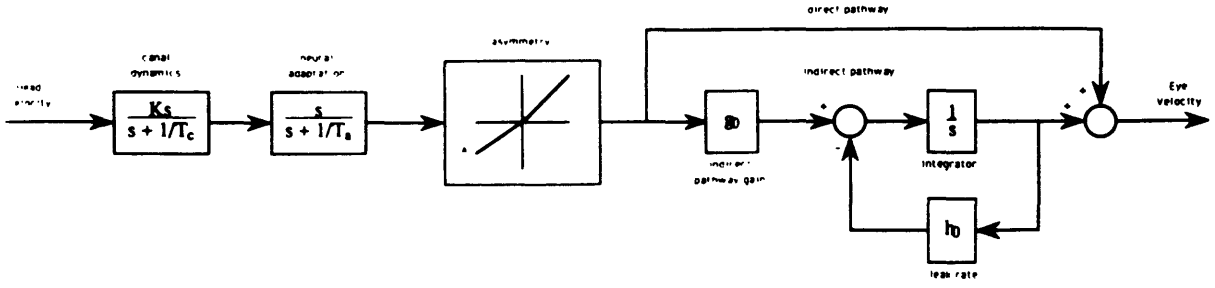


Figure 2.8. Asymmetric Laplace transfer function model of vestibular nystagmus, based upon the Raphan-Cohen model, with no visual input. A directional asymmetry term, A, has been added between the canal afferents and the velocity storage stages.

2.7 Slow Phase Velocity Calculation

Nystagmus patterns contain a sequence of alternating slow and fast phases. In order to calculate the slow phase velocity, the fast phases must somehow be distinguished from the slow phases. It is extremely cumbersome to manually inspect a large data set to remove all of the fast phases. As such, a number of computer-based algorithms have been developed for fast phase detection and removal.

One type of algorithm is based upon pattern recognition schemes (Anzaldi and Mira, 1975). Such algorithms analyze the eye position signal to look for a reversal in the direction of the eye movement. For these to work, the direction of the nystagmus must be specified at the beginning and whenever it reverses; this requires constant monitoring by an operator, and becomes impractical for sinusoidal or pseudo-random stimulation. Thresholds must also be set for the size of "significant" eye movements, which leave these algorithms highly susceptible to noise.

Several velocity-based algorithms have been proposed (Allum et al, 1975; Baloh et al, 1980). The general premise is that, since fast phase velocities are faster than slow phase velocities, a threshold can separate the fast and slow phases. These generally require knowledge of the direction of the nystagmus, and several user-specified parameters related to velocities and/or time durations. Performance tends to degrade when the slow phase velocity becomes too high or too low, and can be especially poor when the slow phase velocity reverses direction without a change in the stimulus (as in Figure 2.3). Some of these algorithms use a running estimate of SPV to classify the current velocity as a slow or fast phase; this is undesirable since such a scheme is unstable in the sense that results are unpredictable once the algorithm goes wrong.

Massoumnia (1983) expanded upon the algorithms of Michaels (1977) and Allum et al (1975) to develop an algorithm which analyzed the acceleration in addition to the velocity. When the acceleration exceeded a certain threshold, an event was detected which would be later classified as a fast phase, a saccade from rest, or some other event; when the acceleration dropped below a second threshold, it was marked as the beginning or end of the event. A pair of acceleration thresholds is better able to detect the beginning and end of saccades than are the velocity techniques. Once again, the algorithm was highly sensitive to noise, and did not perform well for oblique fast phases.

Merfeld (1990) proposed a multidimensional acceleration-based algorithm. The magnitude of the eye acceleration vector was calculated and compared to a similar pair of acceleration thresholds. The advantage of this technique is that it uses information from all axes of eye movement, and as such is better at detecting blinks and oblique saccades. Fast phase detection was also improved by marking the beginning and end of the fast phases as being a few samples before and after the acceleration fell below threshold.

A completely new approach (AATM) was proposed by Engelken and Stevens (1990) which used non-linear order statistic filters. The inherent supposition is that the eye spends more time in fast phases than in slow phases; therefore, more samples correspond to the slow phase velocity than to the fast phases. A histogram is calculated of the velocity values in a sliding one-second window. Then, the extreme values of the histogram are truncated, with a correction for the skewness of the histogram, and the average value of the remainder of the samples calculated as the slow phase velocity at the midpoint of the window. This is a one-pass approach which performs a combination of fast-phase removal and low-pass filtering on the velocity signal. There

are a number of parameters which can be adjusted by the user for optimal performance, but the published values will yield good results in most cases. This algorithm gains on the other techniques because it does not make any assumptions about the vestibular stimulus and does not require user-specified thresholds. Its performance decreases when the slow phase velocity undergoes a sudden change, or when the slow phase velocity is close to the fast phase velocity; however, these are also failings of previously described algorithms.

A second contribution of Engelken was the proposal of another type of order-statistic filter (PFMH) to filter the eye position record. Electromagnetic noise in the eye position signal has been typically reduced by a low-pass filter, which has the bad effect of rounding the corners of the curves representing the nystagmus. Other types of noise may arise from spike artifacts or quantization in the data acquisition system. The PFMH algorithm predicted values for the eye position at a given time, with separate estimates based upon a number of earlier data points (forward estimate) and later data points (backward estimate). The output of the filter was taken to be the median of a number of these predictions. This effectively eliminated spike artifacts, reduced both electromagnetic and quantization noise, and sharpened the corners of the nystagmus to produce better velocity estimates.

3. Methods

3.1 Experimental Equipment

A series of experiments were conducted on the NASA Spacelab Space Life Sciences 1 (SLS-1) shuttle mission to investigate the physiological changes which occur in humans during and after exposure to weightlessness. One of these experiments (E072 FO2) investigated the changes in the human horizontal VOR response, by using a motor-driven rotating chair (Figure 3.1) to provide yaw stimulation similar to that described in Section 2.4. The major equipment is shown schematically in Figure 3.2.

3.1.1 Motor Assembly

The rotating chair was designed and built by students at MIT (Johnson and Gidney, 1983) and has been used for similar experiments on the Spacelab SL-1 and D-1 missions. The 0.75 hp DC motor was capable of delivering 27 ft-lbs of torque to the chair shaft, thereby permitting smooth and constant rotation at speeds in excess of 200 °/s. A motor controller (Inland Motor Division TPA series) and tachometer were used to provide closed loop control of the motor speed. The motor controller required three-phase 208 V power and 20 amps of current; due to a failure on the mother board, an external power supply was mounted on the chair panel to provide ±15 V power. A transformer was used to convert the AC to the DC required to drive the motor.

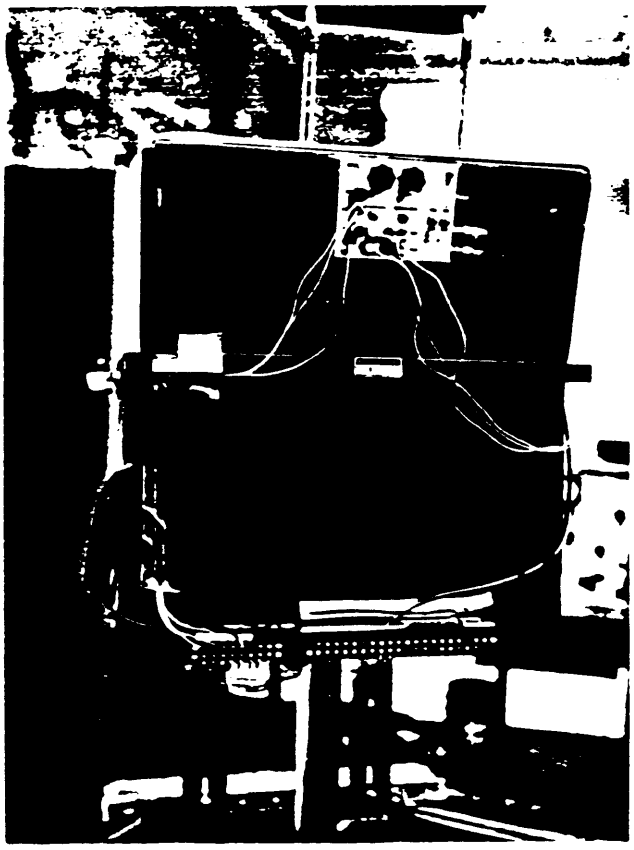


Figure 3.1. SLS-1 rotating chair.

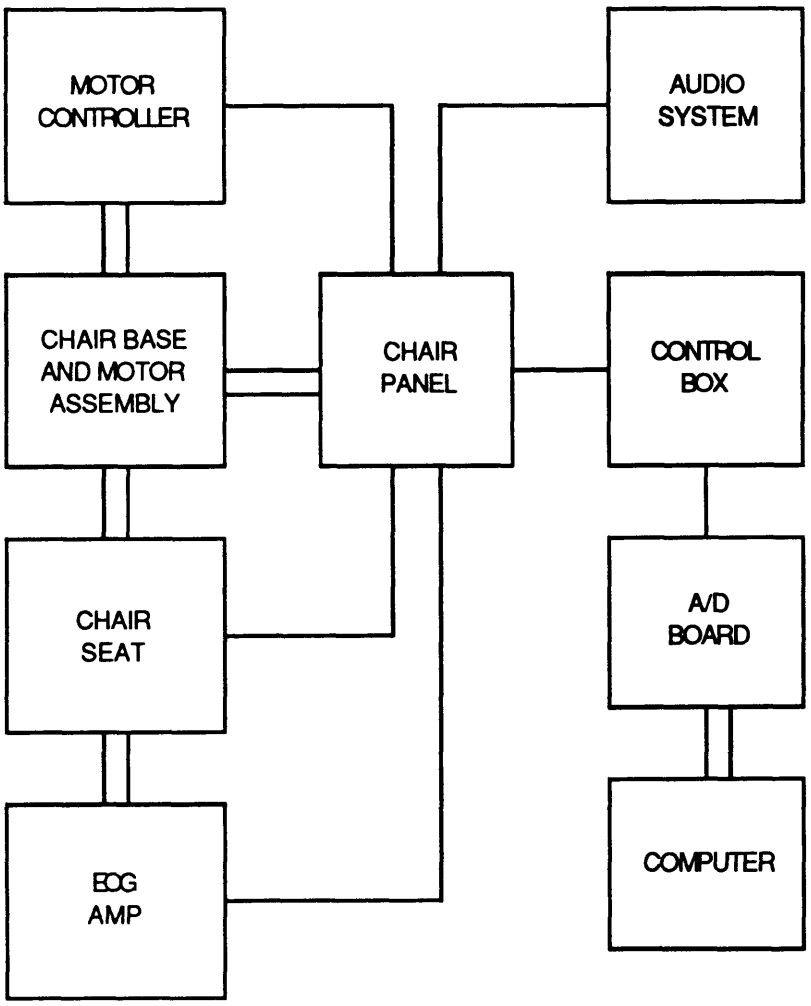


Figure 3.2. Conceptual schematic of experimental equipment. The major hardware components and their interconnections are shown. Adjoining double lines indicate a mechanical connection; a single line indicates an electronic connection.

3.1.2 Safety Features

The subject was positioned upright in a seat atop the chair shaft. For safety reasons, the subject was secured with both a lap belt and straps across the legs. Located on the left armrest of the chair was a switch which was part of the safety interlock circuit for the motor. Similar switches were located on the chair panel and immediately next to the operator of the chair; all three switches must be closed for the motor to function. If the subject were to experience nausea during chair rotation, any of these switches could be thrown to immediately stop the rotation. If a slow stop were desired, a separate velocity control could be used to gradually decrease the speed.

3.1.3 Velocity Control

The velocity command for the motor was generated by analog circuits in the control box, and passed through the chair panel to the motor controller. The operator had control over the direction, speed, and time duration of the chair motion for simple step velocities. Alternatively, an external signal could be input to the control box for complex stimuli such as could be generated by a computer or signal generator. The tachometer on the motor provided a voltage signal corresponding to the actual motor speed; this tach output passed through the chair panel and the control box to a computer.

3.1.4 Event Button

An "event" button was located on the right armrest of the chair, and was to be held by the subject during the experiment for reporting of subjective data. This button was to be pressed for a full second when the subject felt that all sense of rotation had stopped; these times could be compared to the actual zero crossings in the SPV profiles. Since this moment was often hard for the subject to detect, this criterion was modified to specify the moment at which it was impossible to distinguish the direction in which the chair was rotating.

When this event button was depressed, a voltage of 0.87 V was generated. The signal from this button passed through the chair panel to the control box, where it was superimposed on the tach signal.

3.1.5 Sensory Masking

Goggles with interchangeable red and opaque lenses were worn by the subjects. The combination of a low room light level and opaque lenses was sufficient to provide darkness to the subject, removing visual information. Some room light was required so that the operator may watch the subject. The red lenses were used to retain dark adaptation during EOG calibration procedures, since the EOG potential is known to fluctuate with light level (Gonshor and Malcolm, 1971). Red filters were placed over the overhead lights so that dark adaptation could commence as soon as the subjects entered the experiment room; this gave an adaptation time of approximately five minutes before the experiment began. A microphone allowed the investigators to talk to the subject through headphones, while removing any directional auditory cues. Long-

sleeved shirts and long pants were worn (when remembered) to remove wind cues, but this was not consistently done.

3.1.6 EOG Recording

Five neonatal surface electrodes (Wightman W600 pregelled disposable infant ECG & APNEA monitoring electrodes) were used to measure eye position through electro-oculography (EOG). The underlying principle is that the corneo-retinal potential creates a dipolar electric field which is fixed with respect to the eyes; when the eyes move relative to the head, the voltages measured by electrodes fixed to the face will change. Horizontal eye position was measured between electrodes placed on the left and right temples, aligned with the horizontal axis of the eyes. Vertical position was measured between electrodes placed at the top of the right cheekbone and on the forehead above the right eyebrow. The fifth electrode was placed either at the centre of the forehead or on the neck, serving as a reference electrode. The exact location of the electrodes varied slightly across sessions and subjects, but any variability in the voltage readings were reflected in the calibration factors.

Electrode leads connected to a two-channel differential amplifier located on the back of the chair seat. The gain of the amplifier was nominally 6000. The output of the amplifier were two voltage signals (between ±15 V) corresponding to relative horizontal and vertical eye position. Offset controls on the amplifier were adjusted to prevent saturation of the voltage when the electrodes drifted. An external ±15 V power supply connected to the control box, providing power for the amplifier through the chair panel.

These EOG signals passed through sliprings to the chair panel and control box, where they were processed by three cascaded first-order analog low pass filters with corner frequencies of 30 Hz. The filtered EOG signals were then fed to the computer. Variable-intensity light emitting diodes (LEDs) were used to indicate when the voltages approached ±10 V (the range of the A/D board in the computer); this alerted the operator to adjust the appropriate offset control. These analog circuits were powered by another ±15 V power supply which connected to the control box.

3.1.7 Computer

A Macintosh II computer recorded and saved all of the required data. It contained a MacADIOS analog-to-digital (A/D) board capable of digitizing eight analog channels (±10 V) with twelve bits of resolution. Only three channels were used, corresponding to horizontal eye position, vertical eye position, and the combined tach and event button signal.

This data was sampled at 120 Hz using a software package called LabTech Notebook, Version 1.0.1 (Laboratory Technologies Corporation, Wilmington, MA). The data sampling was triggered when the tach signal exceeded 1 V (approximately 60 °/s angular velocity). This threshold was used because it was greater than the voltage of a button push; therefore, an accidental button push could not trigger the data acquisition. The data channels were pre-sampled, and two seconds of data recorded prior to the triggering. Data was sampled for 121 to 123 seconds after triggering. The data was displayed to the operator in real time and saved in a single file on the hard disk.

The sampling, display, and saving of data is controlled by a "setup". An example setup is shown in Appendix B. Due to the erratic behaviour of LabTech Notebook, resulting in computer crashes, the setup occasionally had to be redone in an attempt to increase reliability of the data acquisition. This resulted in different setups being used at different times in the investigation; however, all were functionally equivalent.

3.2 Subjects

The subjects in this experiment were NASA astronauts on the SLS-1 mission. Four subjects were tested both pre-flight and post-flight, and they form the focus of this research. The alternate payload specialist was also tested pre-flight; however, since he did not fly on the mission, he was not tested post-flight and his results will be omitted from this thesis.

Two of the crew members were male, and two female, varying in age from 39 to 47. To maintain confidentiality, these subjects were assigned subject codes M, N, P, and T; all subjects will be referred to as "she".

All subjects were right-handed. All were free of overt vestibular disease. One subject (T) had mild esophoria, reduced acuity in the right eye, and a history of strabismus surgery in childhood. The subjects were restricted from ingestion of caffeine or alcohol for a period of 24 hours prior to each experiment session.

3.3 Experiment Days

Data was collected from the subjects at five sessions prior to the shuttle launch to obtain a baseline for the normal pre-flight response of each subject. The first session occurred fifteen months prior to the launch. Observations from this session suggested modifications in the velocity command and dumping procedure, resulting in the procedure described above. The next four sessions corresponded to approximately 130, 90, 60 and 25 days before launch. For brevity, they were referred to as sessions 2, 3, 4 and 5. Only data from these sessions were included in the pre-flight data set. All pre-flight data was collected at NASA Johnson Space Center in Houston, Texas.

Post-flight data was obtained on five days during a one week period beginning on the day of the shuttle return. Subjects were tested on the day of the return, the day after landing, the following day, two days later, and seven days after landing; for brevity, the five test days were referred to as sessions 6, 7, 8, 9, and 10 respectively. Subjects M, P, and T were tested on sessions 6, 7, 9, and 10. Due to crew time constraints, subject N was not available on the day of the landing; this subject was tested on sessions 7, 8, 9, and 10. Sessions 6, 7, and 8 were referred to alternately as "early post-flight" or "return" sessions, since they were immediately after return to earth. Sessions 9 and 10 were referred to as "late post-flight" or "recovery" sessions, since it was anticipated that the subjects' responses would have almost recovered to their pre-flight norms by the end of the week. All post-flight data was collected at the NASA-Ames Dryden Flight Research Facility in California.

3.4 Experiment Protocol

The basic stimulus consisted of rotating the subject at 120 °/s for a period of one minute, and then stopping the chair. Recording continued for another minute. Due to mechanical constraints, the chair could not instantaneously reach a constant speed of 120 °/s. Therefore, an analog shaping circuit was used to transform the start and stop chair velocity commands to exponential ramps with a time constant of 0.17 s. A second analog circuit was used to control the duration of the spin. Due to variability in the electronic components, this spin duration fluctuated between 59.5 s and 61.4 s. These circuits were part of the control box, and their schematics are shown in Appendix A. The pre-sampling and trigger setup resulted in extra data being recorded before the start of the chair rotation, for a period of time from 1.1 to 1.8 s. This gave a minimum of 59.8 s of data after the chair stop.

Two different types of experiment "runs" were used in conjunction with this stimulus: PRN runs and dumping runs. Together, these two types were referred to as stimulus runs. When the subject was seated with head upright for the full two minutes, this stimulus would elicit horizontal per-rotatory and post-rotatory nystagmus (PRN) as described in Section 2.4; therefore, this configuration was referred to as a PRN run. Dumping runs required the head to be upright for the one minute spin duration, and to be pitched forward 90° subsequent to the chair stop and retained in that position for the full post-rotatory segment. This dumping manoeuvre was different from that used in the SL-1 and D-1 missions. When the chair came to a complete stop, the operator instructed the subject to perform the dumping manoeuvre as quickly as possible. The head was usually stable in the dumped position within two seconds after the chair deceleration had begun. These stimulus runs were performed in low light with the

black lenses in the goggles. During these runs, the subject was instructed to look generally ahead and to not fixate on any imagined point.

The EOG potential was calibrated so that the voltage signals could be converted into degrees of eye movement. This EOG potential depends on the subject, the electrode placement, and the amount of dark adaptation. The calibration target consisted of five black dots placed on the wall in a + arrangement. The angular distance between two adjacent dots was nominally 10°. For a calibration run, the subject placed the red lenses in the goggles and the lights were turned on. The subject was instructed to fixate on each dot for a full second as the dots are called in sequence; the calibration sequence was centre-right-centre-left-centre-up-centre-down-centre. This produced voltage differences corresponding to known angular deviations, and thus the required calibration factors.

It was discovered later that the visual angular distance between calibration points was greater than 10°. The target was constructed with a distance of 50" pre-flight ($51\frac{3}{16}$" post-flight) between the centre of the target and the headrest. However, if one allows for 7" between the retina and the back of the head, the angular distance becomes 11.5°. This discrepancy was only discovered after all of the data analysis had been performed; therefore, the SPV and hence all of the system gains K which are reported in this thesis are underestimated by 15%. A quick inspection of the model (Figure 2.8) shows that K is the only parameter which would be affected by this difference.

Stimulus runs were performed in both clockwise (CW) and counter-clockwise (CCW) directions. A nominal protocol consisted of the following sequence of eleven runs:

01. calibration #1
02. CW PRN #1
03. CCW PRN #1
04. CW dumping #1
05. CCW dumping #1
06. calibration #2
07. CW PRN #2
08. CCW PRN #2
09. CW dumping #2
10. CCW dumping #2
11. calibration #3

Due to the length of the neural adaptation time constant, it was anticipated that there may be residual effects which lasted beyond the two minute run time. Therefore, the directions of the stimulus runs were deliberately alternated so as to reduce buildup of this effect.

In almost all cases, electrodes were placed on the subjects more than thirty minutes prior to the beginning of the test session. In the few cases where this was not possible, the electrodes were given at least ten minutes to allow electrode offset potential drift to settle prior to the beginning of the protocol described above.

When the subject entered the testing room, she was immediately seated in the chair and the seat and leg straps were fastened. Then the electrode leads were connected, and the goggles and headphones put on. The subject was then given a briefing about the

experiment protocol, subjective reporting instructions, safety features, et cetera. This provided sufficient time for the electrodes to settle and for the eyes to begin to dark adapt. Then, the eleven-run protocol was begun. The subject was required to interchange red and black lenses between calibration and stimulus runs. The entire experiment session was allocated a half hour of time.

The combination of four subjects, eight sessions, and eleven runs yielded a grand total of 352 runs. Each of these runs were given a unique identifier, henceforth referred to as a "run code", consisting of the subject letter followed by the session number and the run number. For example, N204 corresponded to the first CW dumping run for subject N on session 2.

Not all of the data runs were successfully completed. Due to time constraints, runs sometimes had to be omitted from the end of the sequence. Other runs were lost due to computer crashes, operator mistakes, or the inability of the subject to continue. The status of completion of all runs is tabulated in Section 5.1.

4. Data Analysis Procedure

A major portion of the work on this thesis was the development of a data analysis procedure which would take the original raw data files and produce quantitative results. The stages of the analysis were to be as automated as possible; this not only minimized the amount of user time and effort required, but also removed individual biases amongst operators. A flow diagram of the overall procedure is shown in Figure 4.1.

4.1 Format Conversion

LabTech Notebook saved the three channels of information in a single file as a matrix, where each row corresponded to synchronous samples, and the columns corresponded to separate time series. One file contained the horizontal eye position (EOG H), vertical eye position (EOG V), and the combined tachometer and event button reading (TACH+EVENT) from a single run. Due to the limitations of LabTech Notebook, all of these files were originally saved under the same name, in different folders, during the experiment session. The names of the files were later altered to unique identifiers referred to as "run codes". The run code consisted of the subject letter, the session number, and the sequence number of the run within the experiment protocol; for example, N204 corresponded to the first CW dumping run for subject N on session 2.

Due to the various setups used at different times, the format and order of these channels varied. For the first four sessions, horizontal eye position (EOG H) was stored in the first column, vertical eye position (EOG V) in the second, and tachometer output (TACH+EVENT) in the third column; this data was saved as four-byte real voltages

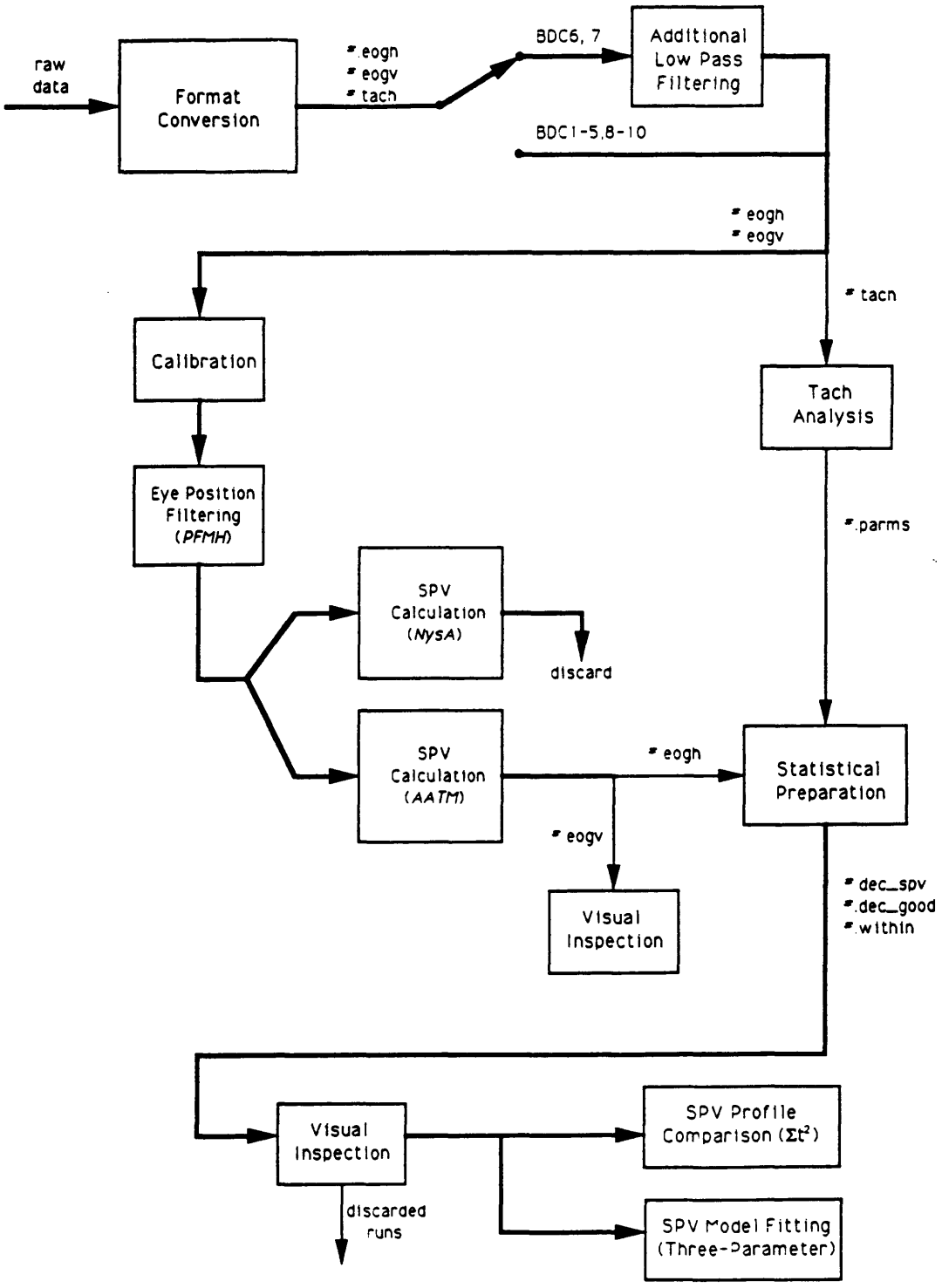


Figure 4.1. Flow chart of overall data analysis procedure.

between -10 V and +9.995 V. In the fifth session, the order of the channels was EOG V, EOG H, and TACH+EVENT; the data was saved as four-byte real voltages. During the post-flight sessions, the order of the columns reverted to the original setup, but the data was saved as two-byte integers between -2048 and +2047, where one unit corresponded to a voltage difference of 4.88 mV.

The majority of the data analysis is done in MatLab 1.2c, a scientific and mathematical software package (The MathWorks, Inc., Natick, MA). MatLab allows the user to write programs called "scripts" which call a wide variety of built-in functions including digital filtering, graphics, and optimization. A C program was written in THINK C 4.0 (Symantec Corporation, Cupertino, CA) to separate the three channels of information into MatLab compatible files. A MatLab file contains a matrix of binary or ASCII data, preceded by a header which indicates the size of the matrix, the format of the data, and a variable name to be associated with the data. The C program assigned the variable names "eogh", "eogv", and "tach" to the three channels of data; each channel was saved as two-byte binary integer data (values between -2048 and +2047) in a separate file which was named by the run code and variable name (e.g. N206.eogh). Due to the different formats in which the data was originally saved, there are three versions of this data conversion program: *BDCF_convert.c* for the first four sessions, *BDC5_convert.c* for the fifth session, and *post_convert.c* for the post-flight sessions. The source code for *BDCF_convert.c* is included in Appendix C; the other versions are merely minor modifications of this program and as such are not included.

In approximately 3% of the runs, the converted data was saved with improper header information. Extensive debugging determined that the source of these errors was not in the C code itself, but in either the compiler, the linker, or the operating system. Since these could not be easily modified, a second C program (*edit_header.c*) was created to

allow the user to correct these errors. Source code for this program is also included in Appendix C.

4.2 Additional Low Pass Filtering

During the first two post-flight sessions, an unusually large amount of 60 Hz electromagnetic interference (EMI) was present in the testing room. The resultant noise on the EOG and tach signals was on the order of 2 V in magnitude. To eliminate this noise, the data from these two days were passed through a fourth-order Butterworth low pass filter with a 24 Hz corner frequency. The MatLab *filtfilt* function was used to produce zero phase distortion. Figure 4.2 shows plots of a section the unfiltered and filtered horizontal eye position from N702. Also shown is a corresponding trace from N502 to illustrate the unfiltered signal quality from the pre-flight test session for the same subject and same stimulus.

This noise problem was solved for the subsequent post-flight sessions by using an isolation transformer. This returned the noise on the signals to the same level as was present during the pre-flight sessions.

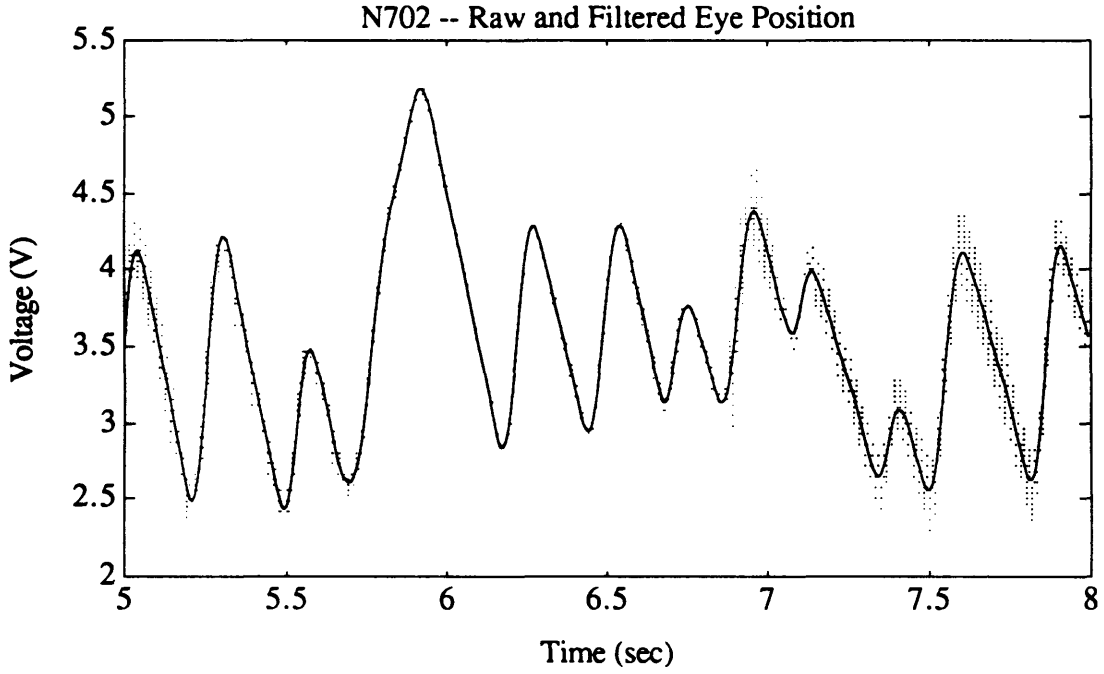


Figure 4.2a. Example of filtered and unfiltered eye position from return session (N702). Dotted line is raw unfiltered eye position, and solid line is filtered eye position as obtained by low pass filtering the data.

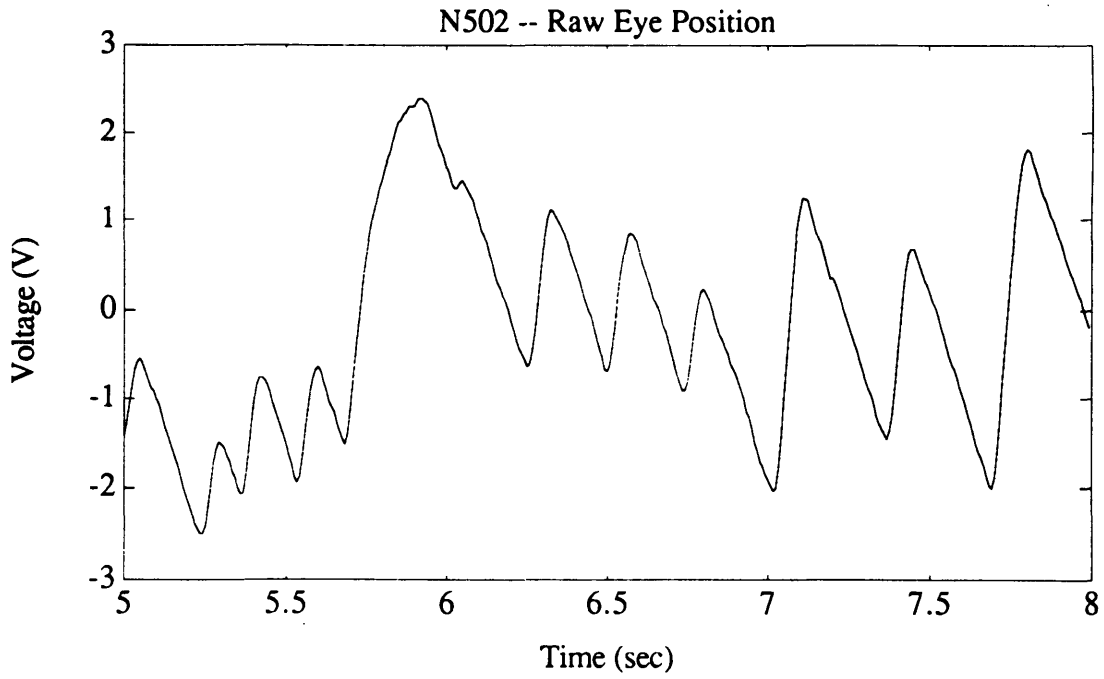


Figure 4.2b. Example of unfiltered eye position from pre-flight session (N502).

4.3 Nystagmus Analysis (NysA) Package

NysA is functionally a set of MatLab scripts for which were written to calculate the slow phase velocity (SPV) envelope from the eye position information. The source code for these scripts is included in Appendix D. The algorithm for saccade detection is essentially that of Merfeld (described in Section 2.7), with some minor modifications. A user's manual for NysA was written and is included in Appendix E.

4.3.1 EOG Calibration

One section of NysA determines the calibration factors to convert from A/D units to degrees of eye movements, using a three-point procedure. The horizontal eye position is displayed, and the user is asked to select the portions of the trace which correspond to fixation upon the right dot in the calibration target. The mean value across all specified intervals is used as the number corresponding to a 10° deflection (amplitude is specified by user). The left and centre dots are similarly specified. The calibration factor, in degrees per unit, is calculated as the ratio of the difference in angular displacements (20°) to the difference in digital units between the right and left targets. This is a simple linear fit to the data which has been shown to be valid for horizontal eye movements. The centre dot is used only to determine an offset value which generally is not very useful for EOG data due to drift. The calibration factor for the vertical eye position is calculated similarly, using the top and bottom dots. While vertical eye movements have been shown to exhibit a non-linear calibration function, a linear approximation is valid over a small range (±30°). Since yaw rotation and dumping manoeuvres are unlikely to elicit significant vertical nystagmus, this linearization should suffice.

This calibration procedure was performed on each of the three calibration runs for each subject in each session. Eye position data for a representative calibration run is shown in Figure 4.3 together with the calculated mean values for each fixation point. The three horizontal and vertical calibration factors from a single session were weighted to determine the calibration factors for each stimulus run as follows:

$$C_2 = \frac{7}{8} C_1 + \frac{1}{8} C_6$$
$$C_3 = \frac{5}{8} C_1 + \frac{3}{8} C_6$$
$$C_4 = \frac{3}{8} C_1 + \frac{5}{8} C_6$$
$$C_5 = \frac{1}{8} C_1 + \frac{7}{8} C_6$$
$$C_7 = \frac{7}{8} C_6 + \frac{1}{8} C_{11}$$
$$C_8 = \frac{5}{8} C_6 + \frac{3}{8} C_{11}$$
$$C_9 = \frac{3}{8} C_6 + \frac{5}{8} C_{11}$$
$$C_{10} = \frac{1}{8} C_6 + \frac{7}{8} C_{11}$$

where C_i corresponds to the calibration factor from the *i*th run. This essentially weighted the calibration factors with respect to the time difference between the calibration runs on either side of a stimulus and the midpoint of that stimulus run.

If one of the stimulus runs was not performed, the weighting factors were changed accordingly. For instance, if the last dumping run was not done, there would be no C_{10}, and the last three calibration factors would be calculated as

$$C_7 = \frac{5}{6} C_6 + \frac{1}{6} C_{11}$$
$$C_8 = \frac{1}{2} C_6 + \frac{1}{2} C_{11}$$
$$C_9 = \frac{1}{6} C_6 + \frac{5}{6} C_{11}$$

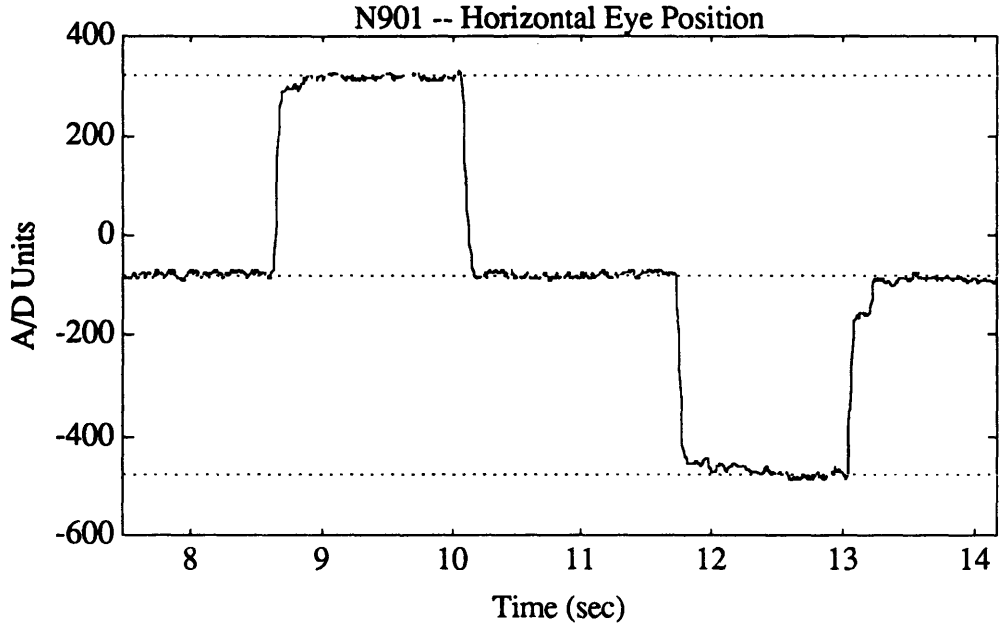


Figure 4.3. Eye position data for a representative calibration run. Solid line is raw eye position in analog-to-digital (A/D) units. Dotted lines indicate, from top to bottom, average eye positions for positive, zero, and negative deflections.

If the data from one of the stimulus runs was lost, but the run was still performed, then the weighting factors for the other runs remained the same. If the first calibration run was not done, or the data was lost or unanalyzable, the values from the second calibration run were used as the calibration factors for the first four stimulus runs; backward extrapolation of the factors from the last two calibration runs was believed to be invalid. Similarly, if the third calibration run was "bad", the values from the second

calibration run were used for the last four stimulus runs. If the second calibration run was bad, the values from the first and third were time-weighted as

$$C_2 = \frac{15}{16} C_1 + \frac{1}{16} C_{11}$$
$$C_3 = \frac{13}{16} C_1 + \frac{3}{16} C_{11}$$
$$C_4 = \frac{11}{16} C_1 + \frac{5}{16} C_{11}$$
$$C_5 = \frac{9}{16} C_1 + \frac{7}{16} C_{11}$$
$$C_7 = \frac{7}{16} C_1 + \frac{9}{16} C_{11}$$
$$C_8 = \frac{5}{16} C_1 + \frac{11}{16} C_{11}$$
$$C_9 = \frac{3}{16} C_1 + \frac{13}{16} C_{11}$$
$$C_{10} = \frac{1}{16} C_1 + \frac{15}{16} C_{11}$$

The calibration factors which were used are tabulated in Section 5.3.

4.3.2 Calculation of Slow Phase Velocity

In the NysA package, the horizontal and vertical eye position data are differentiated twice with finite impulse response (FIR) digital filters to produce eye velocity and acceleration along each axis. The nine-point velocity filter is a convolution of a three-point differentiator and a seven-point low pass filter with a 10 Hz corner frequency, and was calculated by Massoumnia using the REMEZ exchange algorithm to minimize ringing. The resultant filter is expressed as a z-transform as

$$H(z) = \frac{-.0332z^{-4}-.0715z^{-3}-.0678z^{-2}-.0522z^{-1}+.0522z^{1}+.0678z^{2}+.0715z^{3}+.0332z^{4}}{T\,z^{4}}$$

where T is the sampling period (0.0083 seconds in this case). The frequency response of this filter is shown in the NysA manual which is included in Appendix E. The acceleration is calculated by differentiating the velocity with a simple two-point difference

$$H(z) = \frac{z^{-2} - z^2}{2\,T\,z^2}$$

The fast phase detection algorithm uses the magnitude of the two-dimensional acceleration vector, and two thresholds (*thresh_acc* and *thresh_end*). When the acceleration exceeds *thresh_acc*, a fast phase is detected. The acceleration time series is then searched backwards until three consecutive samples fall below *thresh_end*, to minimize the effects of noise. The beginning of the fast phase is specified as the second of these three samples. The time series is then similarly searched forwards for the end of the fast phase, which is the third consecutive sample which falls below *thresh_end*. The algorithm uses the median of the five samples immediately prior to the fast phase to interpolate across this fast phase; this is an estimate of what the SPV would be if the eye were instead performing a slow phase movement. Figure 4.4 shows a section of the horizontal eye position from N502, and the corresponding velocity, acceleration, and slow phase velocity. Figure 4.5 shows the velocity and slow phase velocity for the entire run.

One drawback to the Merfeld algorithm was that these acceleration thresholds must be specified by the user, and they vary widely with the stimulus and recording technique used. Usually, the user must inspect some representative data and try several different values to determine suitable thresholds. This iterative approach can be quite time-consuming, especially for an inexperienced user. NysA estimates reasonable values for

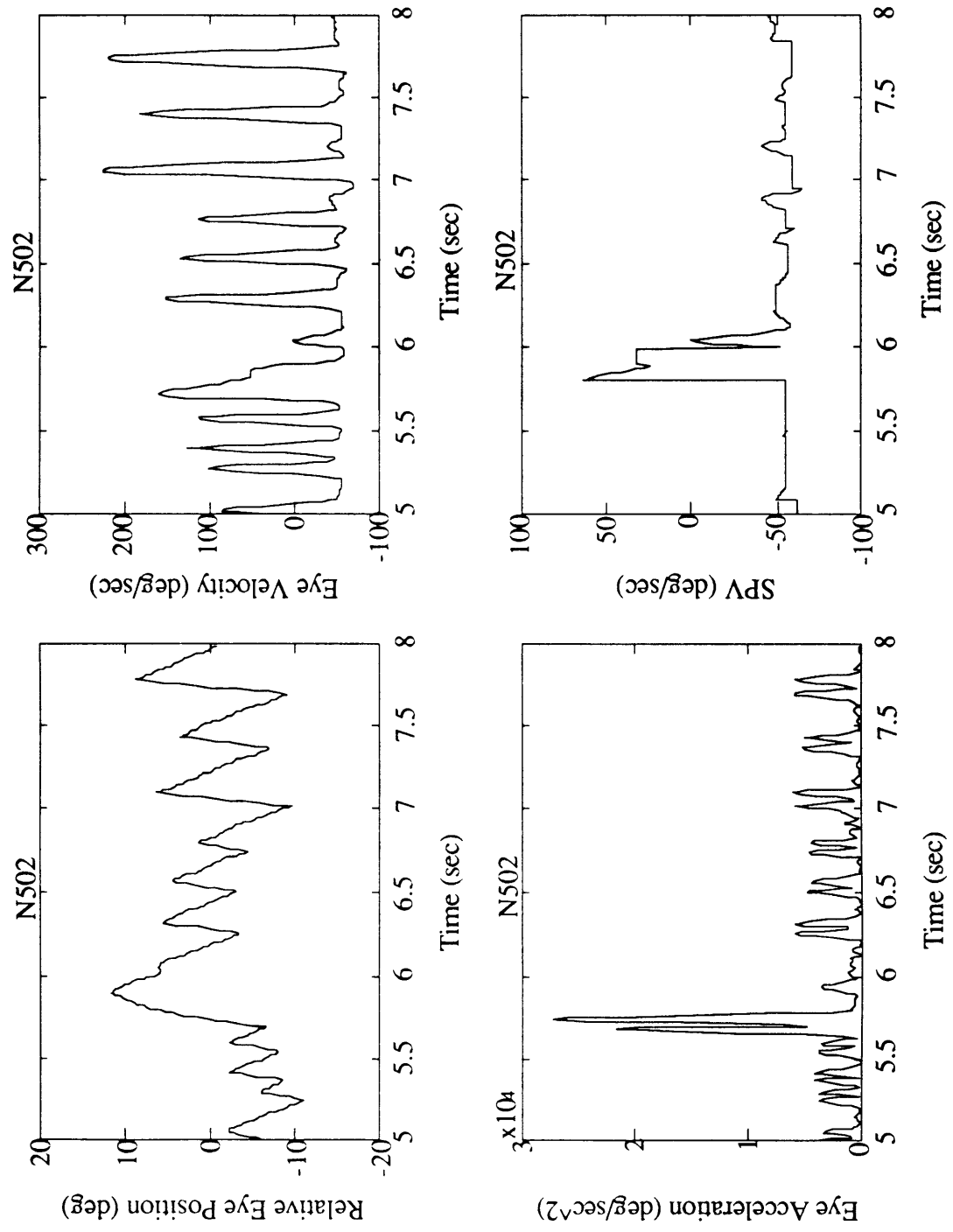


Figure 4.4. Graphical example of NysA performance, showing a section of calculated eye velocity, eye acceleration, and slow phase velocity for a given eye position record (N502). Detection is good except where the velocity contains an inflection point.

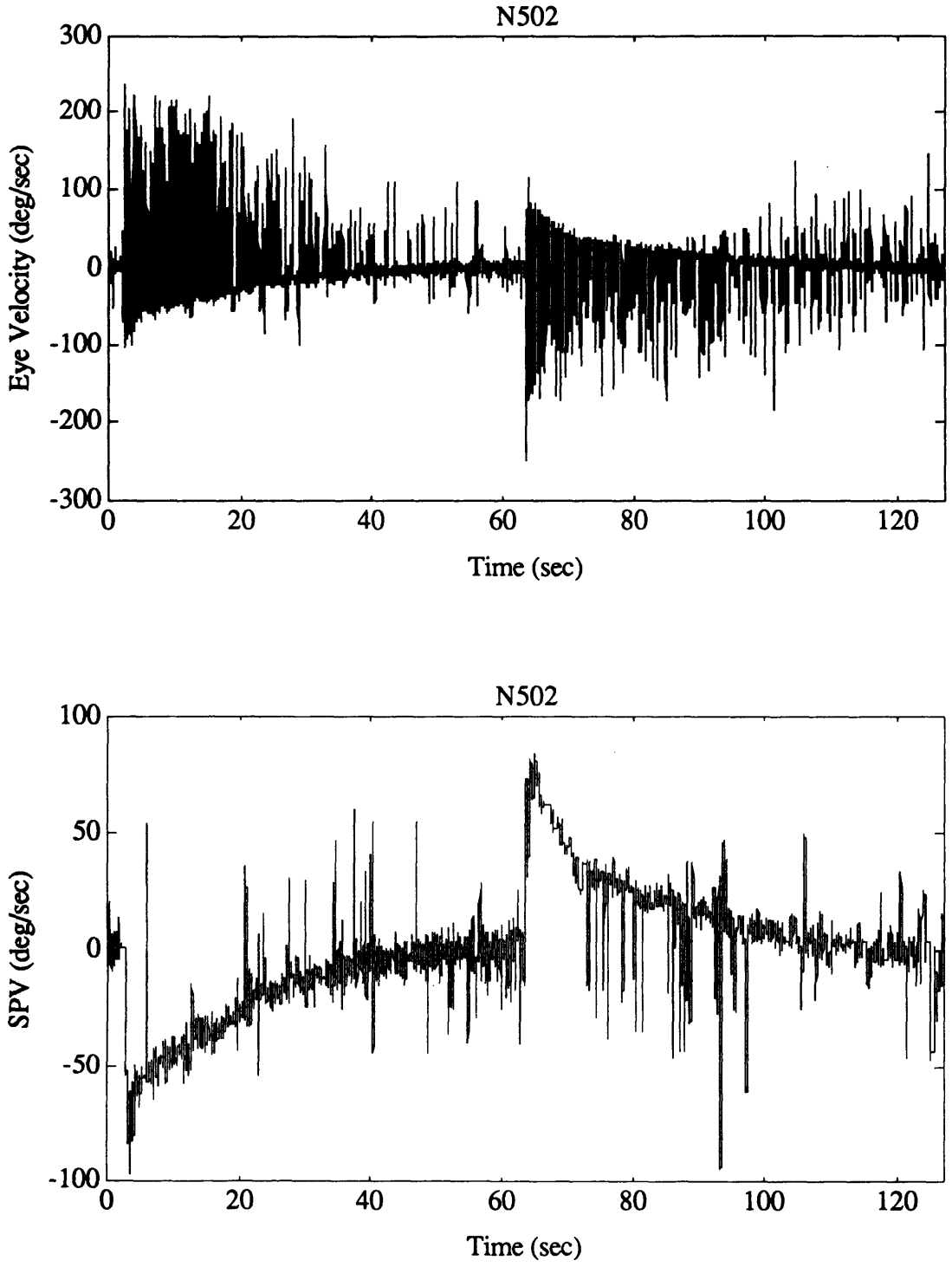


Figure 4.5. Example of NysA performance for an entire run. The top trace shows eye velocity, as calculated by differentiating the raw eye position; the bottom trace shows slow phase eye velocity as calculated by the NysA fast phase detection and removal algorithm. The run shown is the same as in Figure 4.4.

these thresholds by looking at the acceleration magnitude. First, the mean (*mn*) and median (*med*) accelerations are calculated. Then, any accelerations exceeding the sum *mn+med* are temporarily set to *med*; these samples should correspond to fast phases or artifacts. The resulting acceleration contains mostly values corresponding to slow phases. The standard deviation (*sd*) of this acceleration is then calculated to estimate the RMS amplitude of the noise in the acceleration time series. *Thresh_end* is set to *med+sd*, and corresponds roughly to the upper limit of the noise. *Thresh_acc* is set to *mn+2sd* so that any values exceeding this level are almost guaranteed to be fast phases or artifacts. Although no formal analysis of this scheme has been done, the results are generally quite good. The thresholds tend to be 10-20% higher than optimal, but serve as excellent first guesses for the values.

4.3.3 Manual Editing

No fully automated algorithm has been designed which is perfect at detecting fast phases. In fact, clinicians do not necessarily agree with one other when analyzing data by hand. Therefore, the SPV data must be manually reviewed and edited by the user to correct for detection errors. With the modified Merfeld algorithm described above, the beginning or end of a fast phase may be off by a few samples. In some cases, a fast phase may be missed altogether if the acceleration is too small. With the built-in estimation scheme for the acceleration thresholds, the algorithm is about 90-95% reliable, depending on the noise content of the data. However, over two minutes of real data, this may yield dozens of missed fast phases.

The manual editing procedure displays the eye position, velocity, and slow phase velocity to the user. Interpolations from the automatic algorithm may be removed,

modified, or left as they are; missed fast phases can also be specified. Since an interactive user is much better than an automated algorithm at specifying the exact beginning and end of a fast phase, the slow phase velocity is interpolated across these fast phases with a first-order line segment between the last slow phase sample before the fast phase and the first slow phase sample after the fast phase; this is different from the zeroth-order hold which is used in the automated interpolation.

4.4 Order Statistic Filters

An alternative algorithm for calculation of slow phase velocity was proposed by Engelken and Stevens (1990) after NysA had been written. This new method was based upon the use of non-linear order statistic (OS) filters for enhancement of the eye position signal, and for calculation of the slow phase velocity based solely upon the velocity. For this thesis project, a C program (*AATM.c*) was written to implement this algorithm on a Macintosh. The source code for this program is included in Appendix F. A brief description of the algorithm was given in Section 2.7, but more detail is in order here.

OS filters operate on a sliding window, usually of odd length L, and rank-order the data samples; these orderings are the "order statistics". The output of the filter is calculated as a linear combination of these order statistics. When applied to sampled data, the output value of the filter is usually assigned as the middle sample of the window, and the window slides over the full length of the time series; the output values for the first and last (L-1)/2 samples in the time series are thus undefined. A classic example of an OS filter is one which calculates the median of a set of data. The

data are sorted, the middle value is assigned a weight of 1, and the rest of the samples are assigned weights of 0.

An important aspect of some OS filters is the existence of "root" signals, which are invariant under application of the filter. The application of such a filter reduces the magnitude of the differences between the input signal and a root signal. Repeated application of the filter would eliminate these differences, yielding only the root signal.

4.4.1 Eye Position Filtering

Heinonen and Nuevo (1987) introduced predictive FIR mean hybrid (PFMH) filters, which are OS filters whose root signals are piecewise continuous polynomials. They operate on a sliding window of odd length L = 2N + 1. The first N samples are assigned weights to estimate a value for the centre sample; this value is called a "forward prediction" since a series of values are used to predict forward in time. The last N samples in the window are assigned the same weights, in reverse order, to obtain a second estimate (the backward prediction). The formulas used are of the form

$$\hat{x}_F^r(n) = \sum_{i=1}^{N} \left\{ h_r(i) \cdot x(n-i) \right\}$$

$$\hat{x}_B^r(n) = \sum_{i=1}^{N} \left\{ h_r(i) \cdot x(n+i) \right\}$$

where $\hat{x}_F^r(n)$ and $\hat{x}_B^r(n)$ are the forward and backward estimates respectively and $h_r(i)$ are the coefficients for the FIR predictor of order r and length L = 2N+1.

The coefficients for the first three orders are calculated by

$$h_0(i) = \frac{1}{N}, \quad i = 1,\cdots,N$$

$$h_1(i) = \frac{4N - 6i + 2}{N(N-1)}, \quad i = 1,\cdots,N$$

$$h_2(i) = \frac{9N^2 + (9 - 36i)N + 30i^2 - 18i + 6}{N(N^2 - 3N + 2)}, \quad i = 1,\cdots,N$$

The output of the filter is the median of the forward prediction, the backward prediction, and the original centre value. The root signal for an rth order FIR predictor is a piecewise continuous rth order polynomial. Different window lengths and different orders can be used to predict several values for the centre sample; the output of the filter is then the median of all predicted values and the original centre value.

If the data which is being filtered is a combination of a root signal and additive noise, the effect of these filters is to reduce the noise. Iterative passes of the same filter can be used to reduce the noise further after each pass, until the root signal is left.

Engelken used a set of these filters to process nystagmus data, employing three first-order FIR predictors of different lengths and one second-order predictor. For the purposes of our research, two first-order predictors of lengths 13 (N = 6) and 21 (N = 10) were used, and two passes were made on the data. The root signals were therefore straight line segments corresponding to the fast and slow phase portions of nystagmus. The values for N were based upon the expected duration of a fast phase (50-100 ms).

Figure 4.6 shows a section of raw horizontal eye position data and the filtered output. The filtered eye position was shifted up by fifty A/D units so that the details of the two traces could be better seen. The PFMH filtering not only reduced the noise in the signal, but also sharpened the corners of the nystagmus where the eye movements switch from slow to fast phase; these were rounded off by the analog filtering stages. The benefit of the PFMH filtering method was obvious enough that it was also implemented in MatLab as part of NysA, despite the cost in execution speed.

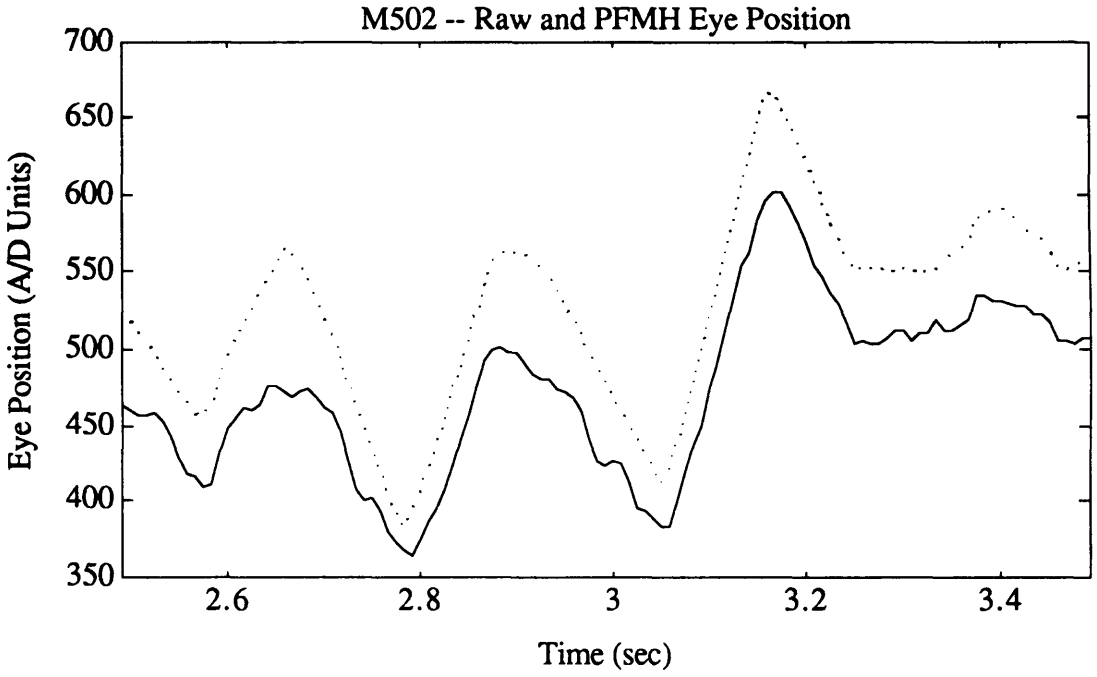


Figure 4.6. Example of performance of PFMH eye position order statistic filter. Solid line is one second of raw eye position (M502); dotted line is filtered eye position, displaced by 50 A/D units so that the differences between the traces can be better seen. Without this displacement, the two traces would overlay each other.

4.4.2 Calculation of Slow Phase Velocity

The eye velocity is calculated by differentiating the eye position with the same digital filter as is used in NysA. A second type of OS filter, an adaptive asymmetrically trimmed-mean (AATM) filter, is then used to estimate the SPV based upon the velocity. This filter type was introduced by Hogg (1974) for use with asymmetric distributions, but without any specific applications. Engelken designed an AATM filter specifically for analysis of nystagmus data with the basic assumption that the eye spends proportionately more time in slow phases than in fast phases. If a histogram is made of a section of eye velocity data, the curve will be skewed with the peak occurring around the slow phase velocity, and with a long tail into the fast phase velocities.

The AATM filter operates on a one-second sliding window of sampled data. The velocities in the window are sorted into ascending order. A simple trimmed-mean (TM) filter "trims" a number of samples from the beginning and end of the sorted sequence and calculates the mean of the remaining samples. This is given by

$$TM_{\alpha} = \frac{1}{L - 2[L\alpha]} \sum_{j=[L\alpha]+1}^{L-[L\alpha]} x_j$$

where L is the window length in samples, α is a user-specified trimming parameter, and x_j is the *j*th ordered sample.

The skewness of the data is estimated from the maximum, minimum, and median values of the ordered samples. To offset the effects of noise, a number of samples are first trimmed from the sequence. The skewness is calculated as

$$S_\beta = \frac{x_{L-[L\beta]} + x_{[L\beta]} - 2 \cdot x_{(L+1)/2}}{x_{L-[L\beta]} - x_{[L\beta]}}$$

where β is a user-specified parameter. This ensures that $-1 \leq S_\beta \leq 1$, where a value of 0 would correspond to a perfectly symmetric distribution.

The AATM filter is similar to the TM filter except that an unequal number of samples are trimmed from the beginning and end of the sequence. The difference in numbers of samples is given by a shift index $K = [MS_\beta]$, where M is the maximum permissible value for K. In the follow-up paper (Engelken et al, 1991), the shift parameter $\gamma = M/L$ was defined as a dimensionless analogy to α and β. The estimate for the SPV at the centre of the window is the output of the AATM filter, given by

$$\hat{x} = \frac{1}{L - 2[L\alpha]} \sum_{j=[L\alpha]+1+K}^{L-[L\alpha]+K} x_j$$

The values which Engelken chose for his parameters were L = 129, α = 0.44, β = 0.12, and γ = 0.186. For the data in this thesis, the sampling rate was 120 Hz, which changes L to 121. Since the speed of rotation on SLS-1 was 120°/s, the peak slow phase velocities were expected to be on the same order as the fast phase phase velocities. γ was therefore doubled to allow a higher shift index, thereby better capturing the peak velocities.

It should be emphasized that AATM does not actually attempt to detect fast phases and attempt to interpolate across them, like most of the algorithms described in Section 2.7. Instead, it estimates the SPV at a given point based upon the distribution of neighbouring values. This estimate is unlikely to be equal to the eye velocity, even at the middle of a slow phase. Essentially, AATM tracks the envelope of the velocity information.

4.4.3 Evaluation of OS Filters

The obvious potential advantage of Engelken's AATM approach over NysA is that manual editing would generally not be required; this removes not only a significant amount of time and frustration, but also operator bias in the editing process. The data was analyzed by both NysA and Engelken's OS filtering method, and the two algorithms were qualitatively compared.

Figures 4.7a through 4.7c show the raw eye velocity for a run (N202) for which the performance of NysA was quite good, and the slow phase velocity as calculated by both methods. The quality of the nystagmus was excellent, with no dropouts, and the SPV followed the expected curve. NysA missed a number of few fast phases, which could be easily removed by manual editing; yet, AATM removed them with ease. There is less noise in the AATM SPV than in the NysA SPV because the trimmed-mean method also serves as a moving averager.

Figures 4.8a through 4.8c show the raw eye velocity for a run (T507) for which the performance of NysA was not very good, and the slow phase velocity as calculated by

both methods. There were frequent and extensive dropouts in the nystagmus, and a lot of fast phases were missed by NysA. However, the SPV calculated by AATM properly tracked the velocity envelope, with significantly long dropouts indicated by a reduced SPV.

The first four pre-flight sessions were analyzed by both NysA and AATM, and the SPV profiles were compared with the velocity signal and with each other to determine the performance of the two algorithms. Overall, the performance of AATM was substantially superior to that of NysA. Therefore, the AATM method was used to produce the SPV profiles for the later statistical analysis.

The only portions of the profiles where the AATM method did not work very well were near the start and stop of the chair. Here, the SPV abruptly changed by a large level (about 80 °/s); the inertial effects of a one-second sliding window tend to produce poor SPV estimates for the first one-half second after the change. More importantly, for the first few seconds after the start and stop, the nystagmus beat frequencies were as high as eight per second, and the fast phase velocities were frequently as low as 100-150 °/s. When the beat frequency was high and the ratio of fast to slow phase velocities was less than 2, the SPV estimate determined by AATM was generally much lower than that determined by NysA. In most cases, this underestimation only lasted for a period of about 2-3 seconds. However, in some runs, nystagmus was weak enough that the fast phase velocities were under 100°/s for virtually the entire trace. The AATM SPV estimate appeared to be consistently low throughout the run, but the performance of NysA was no better.

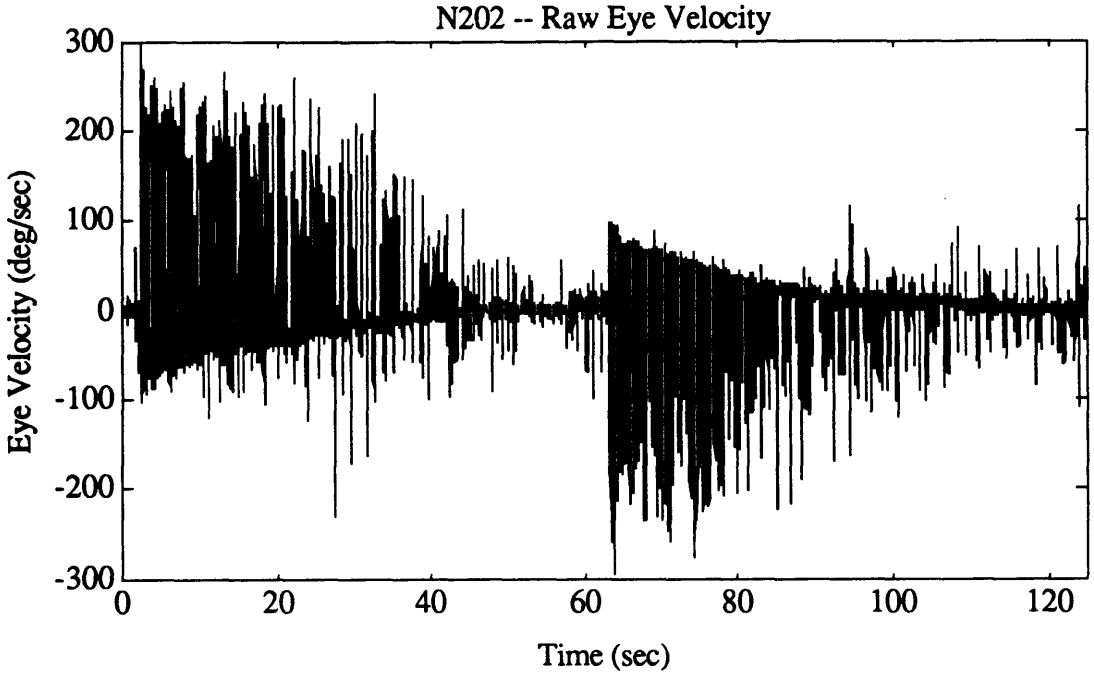


Figure 4.7a. Eye velocity for an excellent quality run (N202).

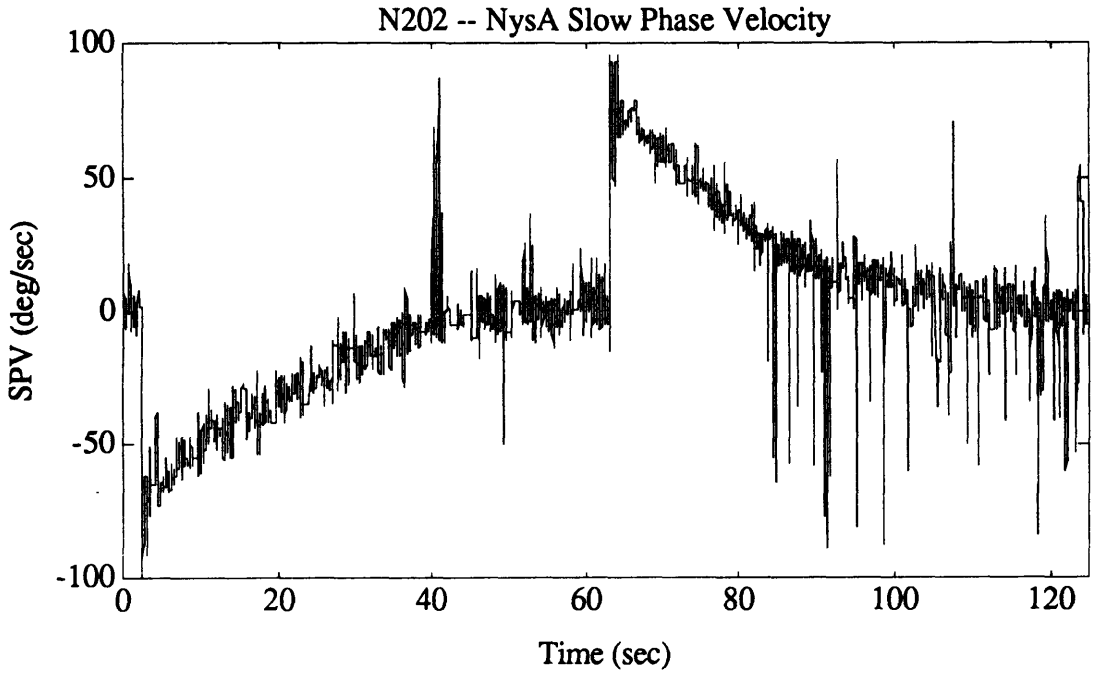


Figure 4.7b. NysA slow phase velocity for an excellent quality run (N202).

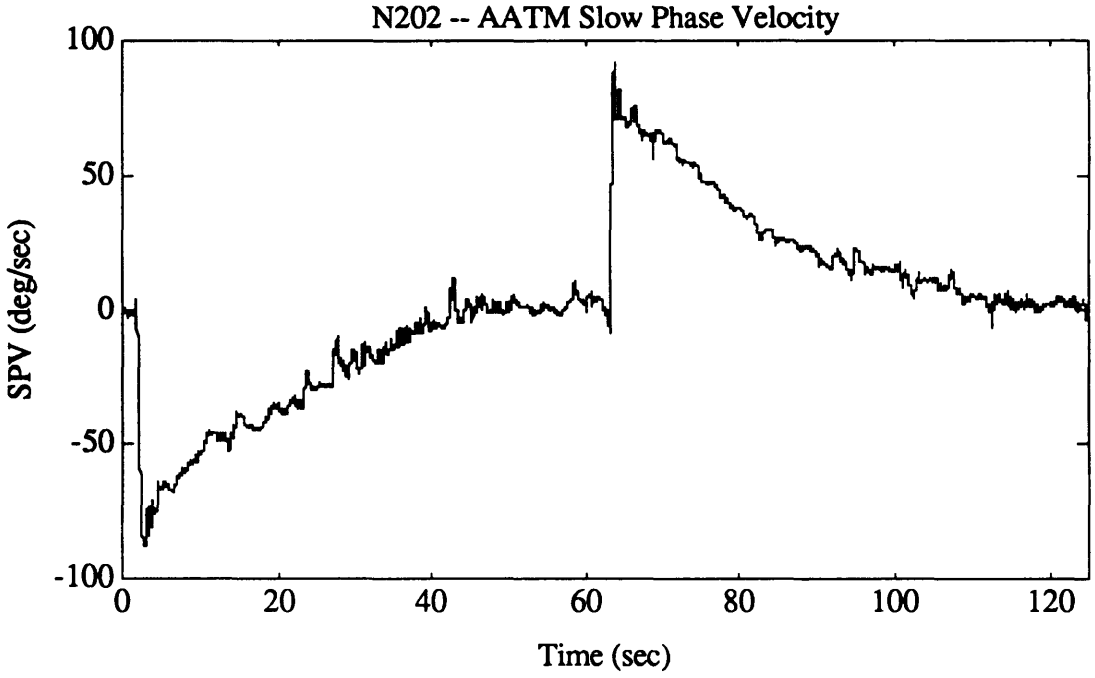


Figure 4.7c. AATM slow phase velocity for an excellent quality run (N202).

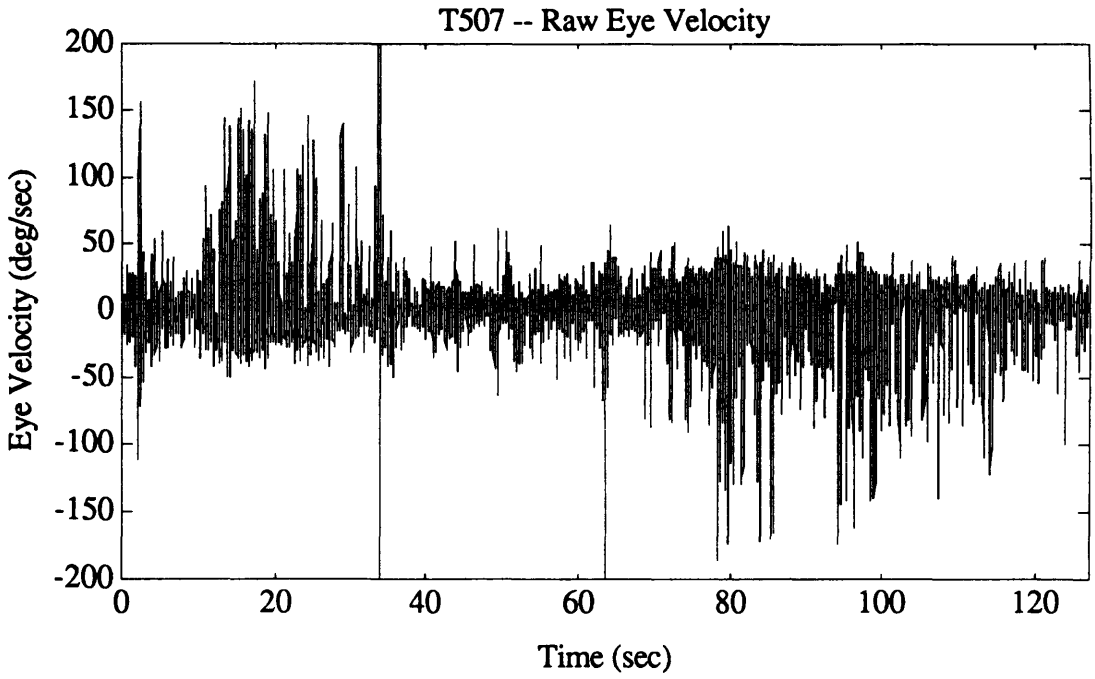


Figure 4.8a. Eye velocity for an intermediate quality run (T507).

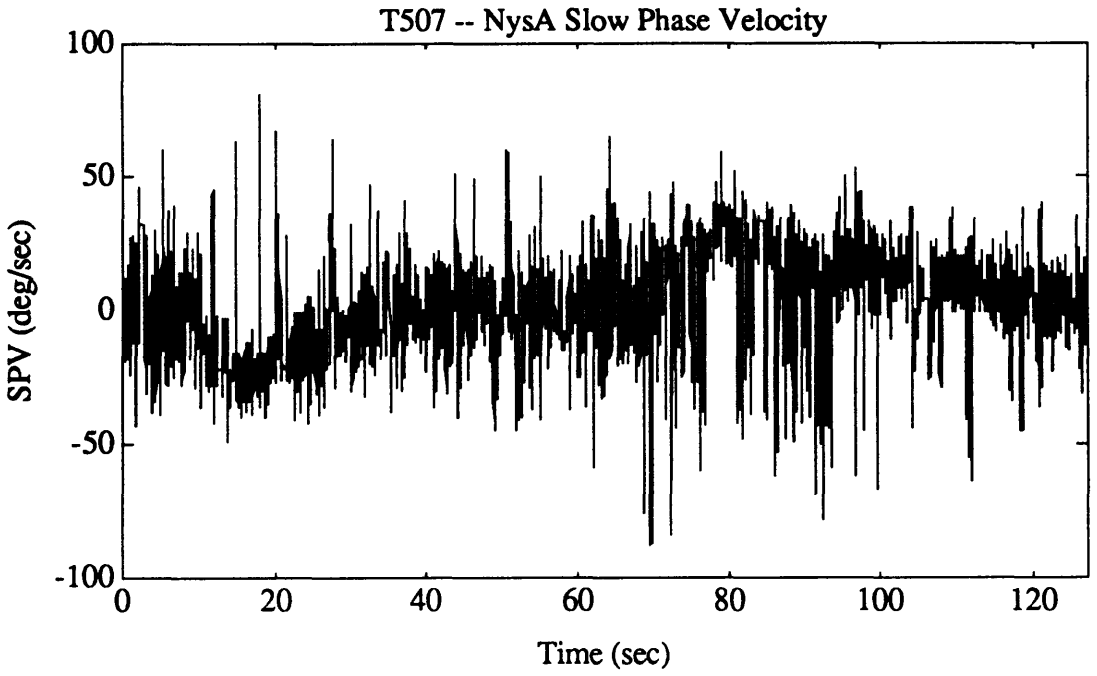


Figure 4.8b. NysA slow phase velocity for an intermediate quality run (T507).

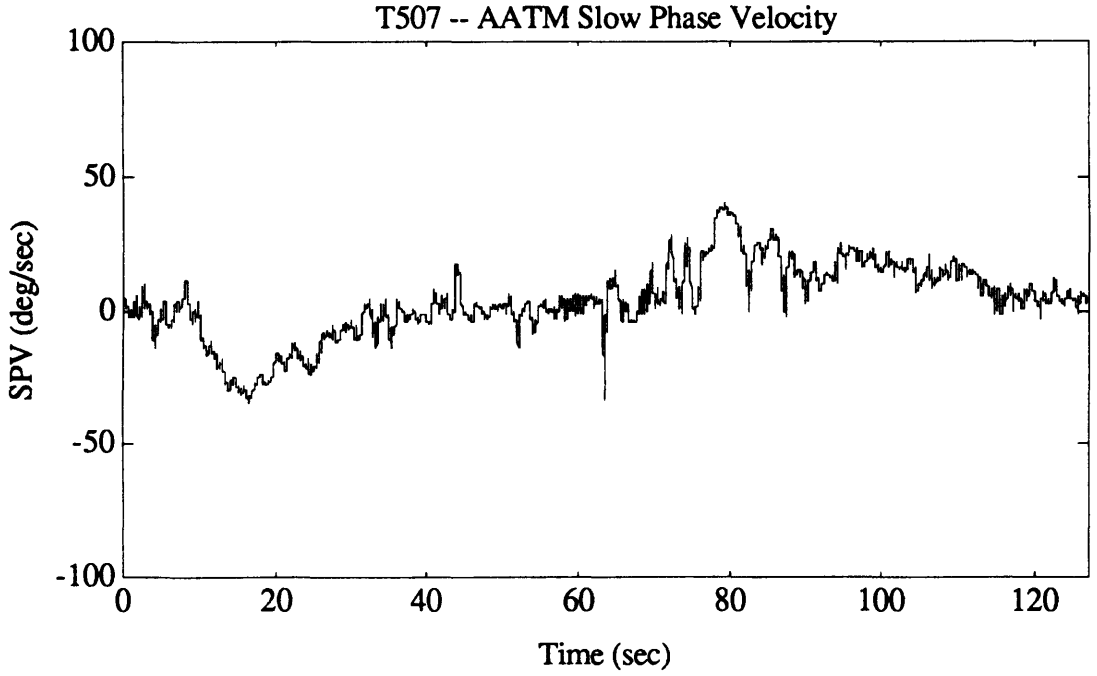


Figure 4.8c. AATM slow phase velocity for an intermediate quality run (T507).

4.5 Tach Analysis and Subjective Reporting

The tach signal contains information about the rotational velocity of the chair. This gives us the exact times of the start and stop of the chair motion, as well as the steady state velocity. If the chair motion smoothly follows the command velocity, this signal should consist of a fast exponential ramp (time constant of 0.17 s) up to a constant speed of 120 °/s for approximately 60 s, followed by a similar exponential ramp down to zero.

Superimposed upon the tach signal is the reading from the event button, which the subject used to report the times at which the sensation of rotation disappeared, during both the per-and post-rotatory sections. The effect of pressing the button is to add a constant voltage to the tach signal until the button is released.

A MatLab script (*tachan*) was written to analyze this tach signal. Figure 4.9a demonstrates an idealized tach signal, and the six parameters which are extracted by *tachan*. Only one button push was considered for each of the per- and post-rotatory sections, since we were interested in the time at which sensation of rotation first disappeared; it was not anticipated that there would be sufficient time for the sensation of rotation to disappear, reverse direction, and disappear again. Figure 4.9b shows an actual recorded tach signal for comparison; notice that it is quite noisy since it is not low pass filtered as are the EOG channels.

The central concept in *tachan* is to look for a large change in the tach signal. This is done in the *delta_tach* script by calculating the mean and standard deviation of one second of data, starting at a specified sample number, and searching forward in time until the sample value deviates from this mean by more than four standard deviations.

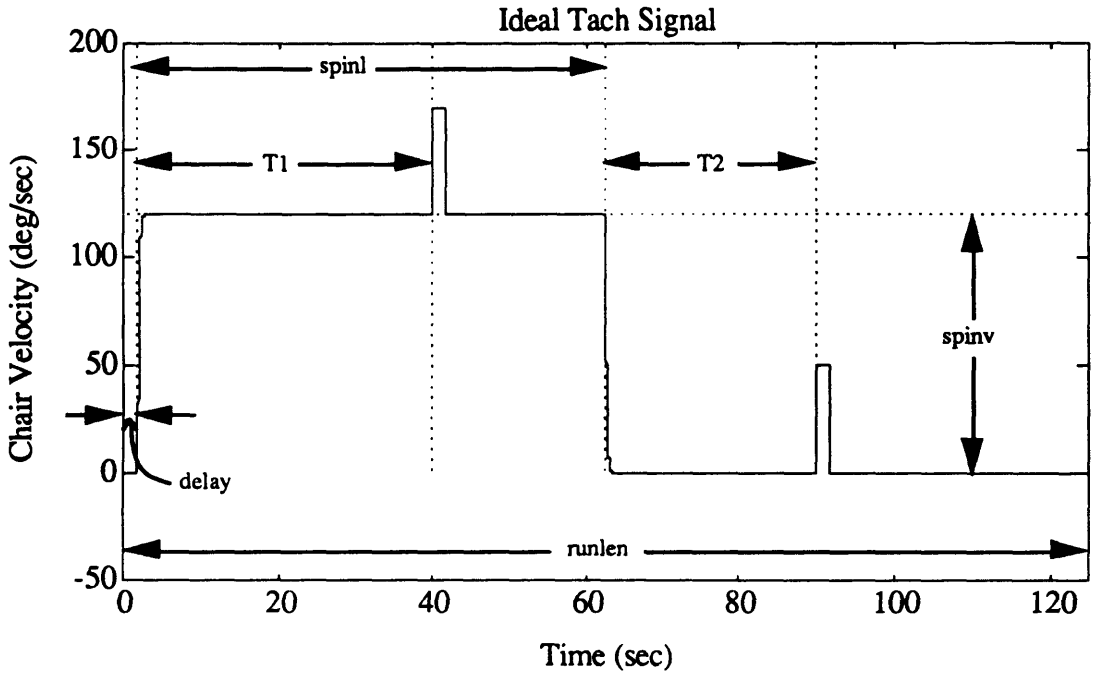


Figure 4.9a. Idealized tach signal showing the parameters which are extracted.

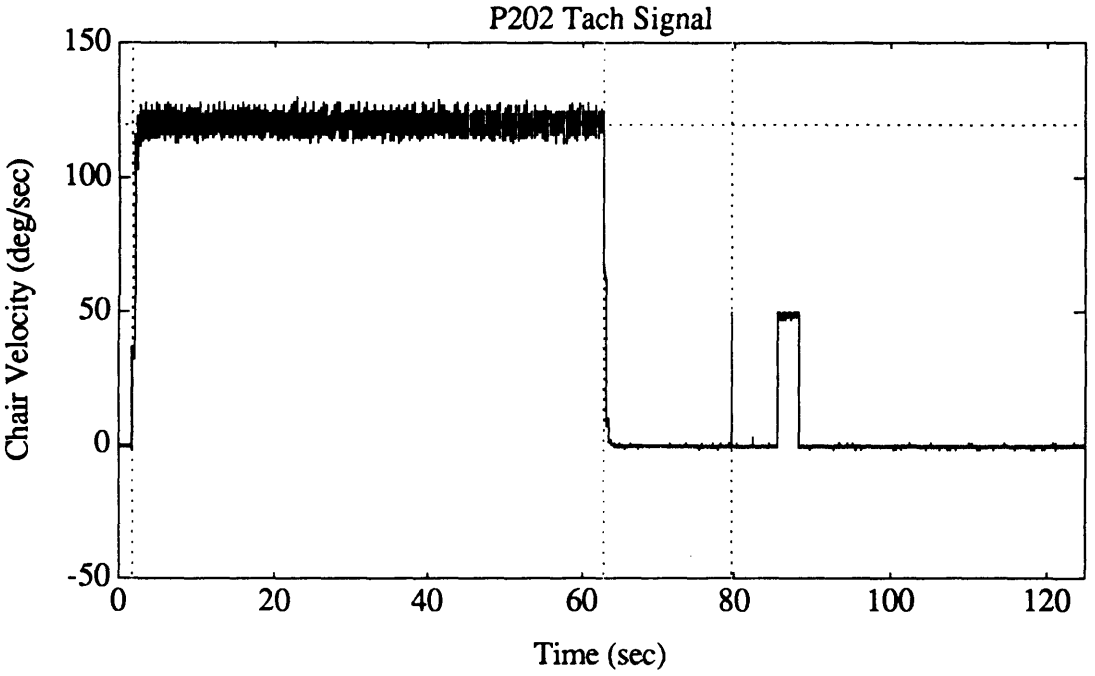


Figure 4.9b. Actual tach signal (P202) showing chair angular velocity, along with superimposed button pushes.

The actual algorithm sets some thresholds and performs some integrity checks on the sample values returned by *delta_tach*. In particular, the algorithm makes use of the fact that the steady state chair speed (calibrated at 2.08 V) was more than double the event button voltage (calibrated at 0.87 V). Also, the shapes of the changes were different; starts and stops were exponential ramps which caused slower changes than the sudden step changes due to button pushes. Small noise artifacts are ignored by only looking for changes as large as 0.5 V.

The start of chair motion is detected by searching for the first large change in the tach signal. *Tachan* checks that the chair has come up to speed one second after the change; if not, it has detected a button push, and continues searching until the start has been found. This determines *delay*.

When the start has been located, the next large change in the tach signal must be either a button push or the chair stop. If the change is a sudden step by an amount exceeding 0.5 V, then a button push has been detected and T_1 is determined as the time between the start of chair motion and the button push; all subsequent button pushes are ignored until the stop is found. If the change is too slow for a button push, then the stop has been detected and there is no per-rotatory button push; in this case, T_1 is set to *NaN*, which is a special MatLab symbol for an illegal value (Not a Number). The time from the beginning of the chair deceleration to the time of the post-rotatory button push (T_2) is detected in the same way as the per-rotatory button push. If no post-rotatory button push is present, T_2 is also set to *NaN*.

The spin velocity *spinv* is calculated as the mean sample value in the interval beginning three seconds after the start of chair motion, and ending five samples prior to the first

button push (if present) or the stop (if no button push). The chair should be at its steady state velocity throughout this period.

The events corresponding to the start, stop, and button pushes are shown in Figure 4.9, along with the steady state chair velocity. Notice that secondary button pushes are ignored. Source code is included in Appendix G.

4.6 Statistical Preparation

There are still three analysis stages which must be performed before statistical comparisons can be made on the SPV profiles. The data from each run must be normalized with respect to time so that the start and stop epochs of all runs coincide. Any dropouts or artifactual data must be removed, since they occur at random times from one run to the next. Finally, the 120 Hz SPV profiles should be decimated to a manageable sampling rate such that it is practical to fit models to the data and use statistical analysis techniques. MatLab scripts have been written to perform these steps, and are included in Appendix H.

4.6.1 Time Normalization

Before SPV time series can be properly compared, they must be "normalized" with respect to time so that the start and stop times correspond from run to run. This also sets a one-to-one correspondence between any two data points occurring at the same time in separate runs. Determine means and variances in responses then becomes trivial, and model fitting can be done more easily.

All samples prior to the start of chair motion, as detected by *tachan*, are discarded. The sample detected as the start of the chair motion is used as the first sample of the new normalized SPV profile.

The variability in the chair spin duration timer may cause it to be slightly longer or shorter than the ideal sixty seconds. If the spin duration is longer, only the first sixty seconds of per-rotatory SPV data are used. Since the spin duration only varies between approximately 59.5 and 61.3 seconds, the slow phase velocity should not be changing appreciably near the end of the spin. If the spin duration is shorter, all of the per-rotatory SPV data is saved, and the median of the last five per-rotatory SPV samples is used to extrapolate to a full minute.

The remainder of the normalized data consists of the sixty seconds of SPV samples beginning at the stop sample. Information more than one minute after the stop is discarded; if there is less than one minute of data after the stop, the median of the last five post-rotatory samples is used to extrapolate to a full minute. All of the runs for this experiment contained at least one minute of data after the stop. Therefore, the normalized slow phase velocity time series contains exactly 14401 samples (two minutes at 120 Hz), the first sample corresponding to the start of the chair acceleration (start time), and the middle sample corresponding to the start of the chair deceleration (stop time).

4.6.2 Outlier Detection

It is well known that a decrease in the level of alertness of a subject will result in a decrease in the VOR gain. This may result in a temporary suspension of nystagmus, here referred to as a "dropout", and the SPV in these sections is approximately zero. On the earlier SL-1 and D-1 missions, the physical and mental demands placed on the crew were so severe that fatigue was a significant problem. When the data from these missions was analyzed, it was manually edited. However, dropouts were not quantitatively defined; as such, some were removed while others may have been missed and averaged in with the rest of the data.

Artifacts may also occur in the SPV data if the electrodes or electrode leads are tugged or if the operator adjusts the EOG amplifier offset. These dropouts and artifacts fall into the general category of "outliers". The corresponding sections of the SPV data were marked as anomalous data to be disregarded for comparison with other curves. A new automated algorithm was developed which includes two tests to detect these outliers.

In this algorithm, a dropout is considered to be a period when the SPV decreases to a near-zero value, when the SPV before and after the period is above zero and the stimulus is constant. If the dropout is short in duration (on the order of 250 ms), the skewing nature of AATM will interpolate across the dropout. If the dropout is slightly longer, the effects of AATM will be to interpolate a somewhat lower SPV across the interval. Hence, a dropout may result in a section in which the SPV is above zero, but fairly low. Since the amplitude of the noise in the SPV data is approximately 10 °/s on average, dropouts are only noticeable if they occur in sections where the magnitude of the real SPV is greater than this noise level.

A sub-section of the SPV data is taken from one second after the chair start, until the average SPV is less than 10 °/s in magnitude, with the condition that at least 20 s of data is taken. The natural logarithm of this data is calculated and fit with a straight line. The first second is excluded since the SPV increases rapidly as the chair rotation starts, and there may be a slight delay in the AATM response at high beat frequencies. This sub-section will be referred to as the "exponential portion" of the curve. The models in Section 2.6 suggest that a simple exponential is not strictly an accurate fit to SPV data such as this. However, dropouts are much larger deviations from an exponential fit than are the real data; therefore, they can be removed with some confidence.

Linear regression is used to determine a linear fit to the log-transformed data (Cheney and Kincaid, 1985). Given a time sequence $\{t_i\}_{i=1}^n$, and a set of values $\{x_i\}_{i=1}^n$ at those times, an optimal straight line function can be fit of the form

$$y = \hat{m}t + \hat{b}$$

by minimizing the sum of square errors given by

$$S(m,b) = \sum_{i=1}^{n} (mt_i + b - x_i)^2$$

Optimal m and b are determined by setting the partial derivatives to zero as follows

$$\frac{\partial S}{\partial m} = \sum_{i=1}^{n} 2(mt_i + b - x_i)t_i = 0$$
$$\frac{\partial S}{\partial b} = \sum_{i=1}^{n} 2(mt_i + b - x_i) = 0$$

This gives two equations in two unknowns for m and b, resulting in the deterministic formulae

$$\hat{m} = \frac{n\sum_{i=1}^{n} x_i t_i - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} t_i}{n\sum_{i=1}^{n} t_i^2 - \left(\sum_{i=1}^{n} t_i\right)^2}$$

$$\hat{b} = \frac{\sum_{i=1}^{n} t_i^2 \cdot \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} x_i t_i \cdot \sum_{i=1}^{n} t_i}{n\sum_{i=1}^{n} t_i^2 - \left(\sum_{i=1}^{n} t_i\right)^2}$$

The root mean square (RMS) error is then calculated as an indicator of the quality of the fit. A dropout sample is defined as any sample whose value deviates from the fitted line by more than three times the RMS value, and whose value is less than two log units (7.4 °/s). This non-zero threshold was decided upon because the SPV should be above this level and because the noise should be at about this level. If a sample value deviates from the fitted line by more than six times the RMS value — Chauvenet's criterion (Huber, 1981) — it is also marked as an outlying sample independent of any threshold. Once these samples have been marked, any sample within half a second of one of these samples is also marked as an outlier, so that the transients at the beginning and end of the dropout will be removed.

This procedure is then repeated on the remaining set of "good" data (the exponential portion with these outliers removed). A new straight line fit is determined, resulting in a new RMS value based on the differences between the good data and the straight line fit. The entire exponential portion, including samples previously marked as outliers, is then compared to this new straight line and RMS value. This will generally detect more

outliers than did the previous iteration, since the straight line is a better fit to the good data, and result in a monotonically decreasing RMS value. In cases of extremely poor data, samples may even be incorrectly marked as outliers by the first pass; this will usually be corrected by the second or third pass. This iterative process continues until the RMS value converges to within 20% of the RMS value from the previous iteration. In over 200 applications of this process, this convergence always occurred; intuitively, it should since the RMS should be monotonically decreasing and non-zero.

If an outlier is detected in the first half second of the exponential portion, this is indicated as an "underflow" in the method such that the dropout extends outside the bounds of the exponential portion. The necessary amount of data before the exponential portion is marked as bad so that a full half second is disregarded both before and after the detected outlier. Similarly, if an outlier is detected in the last half second, an "overflow" occurs and the necessary amount of data after the exponential portion is marked as bad.

This technique is then applied to the corresponding "exponential portion" of the post-rotatory data, from one second after the stop, until the average SPV falls below 10 °/s in magnitude.

While no rigorous analysis has been made of this method, it has converged to a reasonable linear fit within six iterations even for very poor data sets where the percentage of bad data is 67%, and is quite good at marking the beginnings and ends of the outliers. Figure 4.10a shows a section of SPV data from a high-quality run (N207). Figure 4.10b shows the corresponding log-transformed SPV data (dotted line), and the linear fits from successive iterations. Solid lines show the best-fit lines, while dashed lines indicate a band of ±3 times the RMS value. Numbers down the

right side of the graph, adjacent to the best-fit and ±3 RMS lines, indicate the number of the iteration which produced that line. Notice that the second and third iterations yielded very similar best-fit lines, but the third iteration yielded a substantially lower RMS value; this is due to the extra "bad" data points which were removed. Four iterations were required to reach convergence, with the fourth fit being virtually identical to the third. Figures 4.10c and 4.10d show the SPV for the entire run before and after the outliers were removed.

Figures 4.11a and 4.11b show the SPV data for a low quality data set (T507), and the first four iterations of the outlier detection algorithm. Again, the second and third iterations produced nearly identical linear fits; however, this was merely a coincidence and was not at all typical of the overall data set. Although the first linear fit is "pulled down" by the dropouts, the fourth linear fit is almost identical to the fifth and final fit (not shown). Figures 4.11c and 4.11d show the SPV for the entire run before and after the outliers were removed.

This technique detects only artifacts that occur during the "exponential portions" of the SPV profiles. It is quite possible that artifacts occur outside of these portions. Since the magnitude of SPV reversals is not expected to exceed 20 °/s, the remainder of the curve should be less than 30 °/s, even allowing for some noise. Therefore, an outlier sample in these portions is defined as any sample whose value exceeds a simple magnitude threshold, in this case 30 °/s. As in the logarithmic technique, these outliers are extended for a full half second before and after the threshold is exceeded. Underflow and overflow are also handled similarly so as to identify artifacts that extend into the exponential portions.

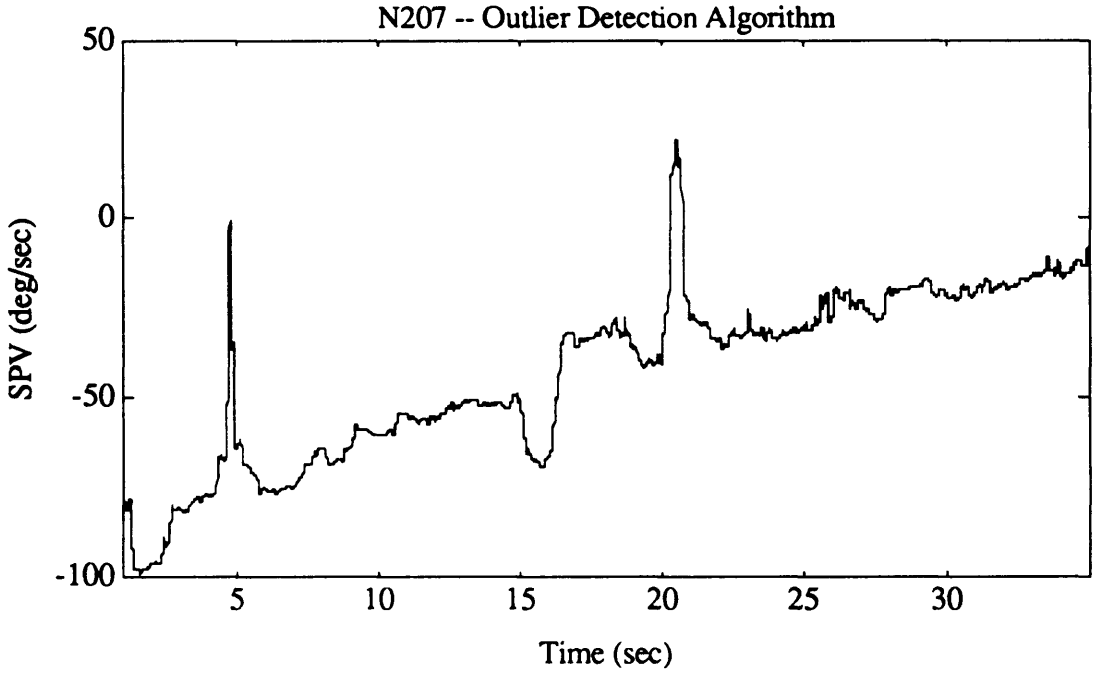


Figure 4.10a. Exponential portion of N207 post-rotatory SPV, as calculated by AATM.

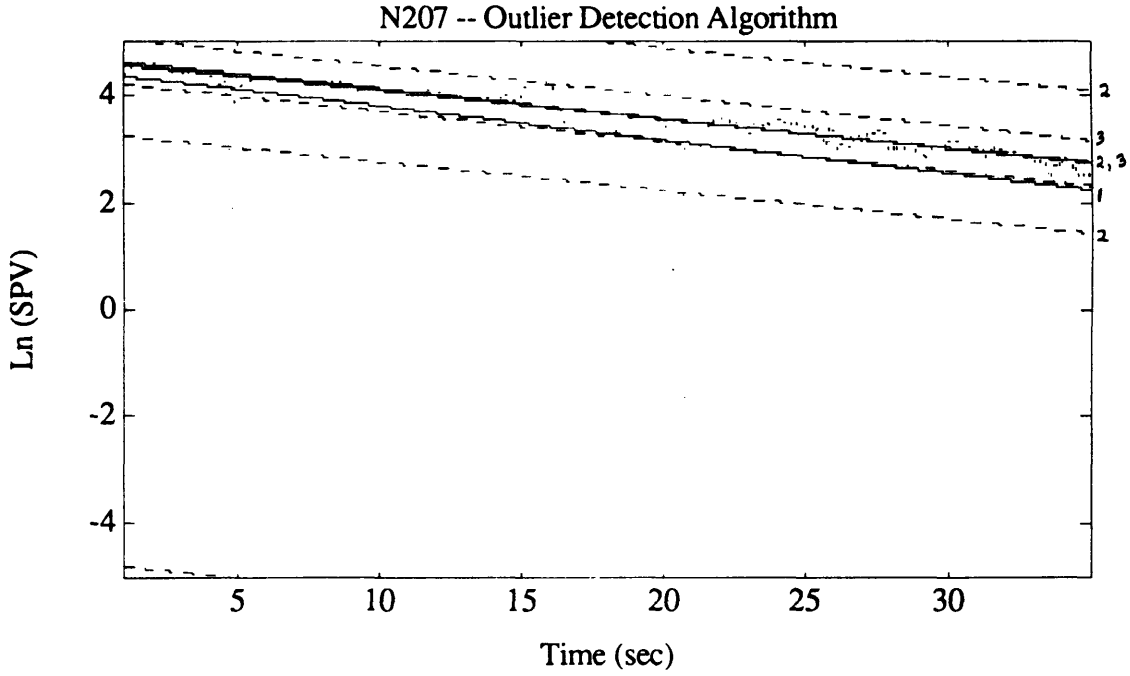


Figure 4.10b. Log-transformed data for exponential portion of N207 post-rotatory SPV, together with first three iterations of outlier detection algorithm. Dotted line is SPV data, solid lines are lines of best fit for each iteration, and dashed lines indicate bandwidths of ±3 RMS values.

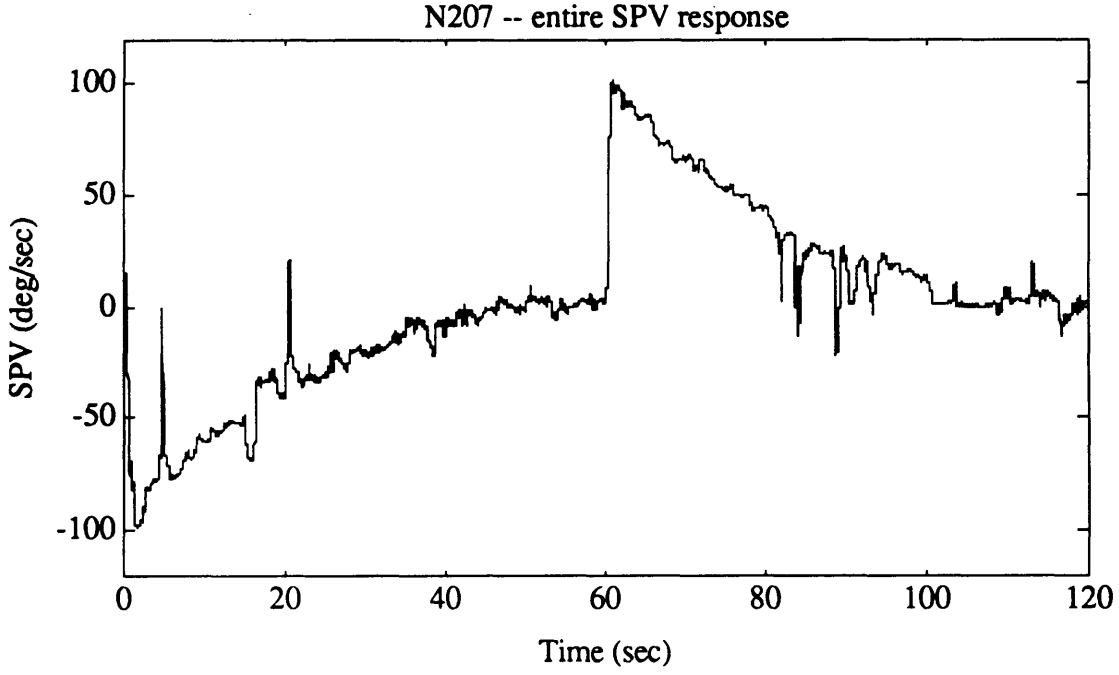


Figure 4.10c. Entire N207 SPV, as calculated by AATM.

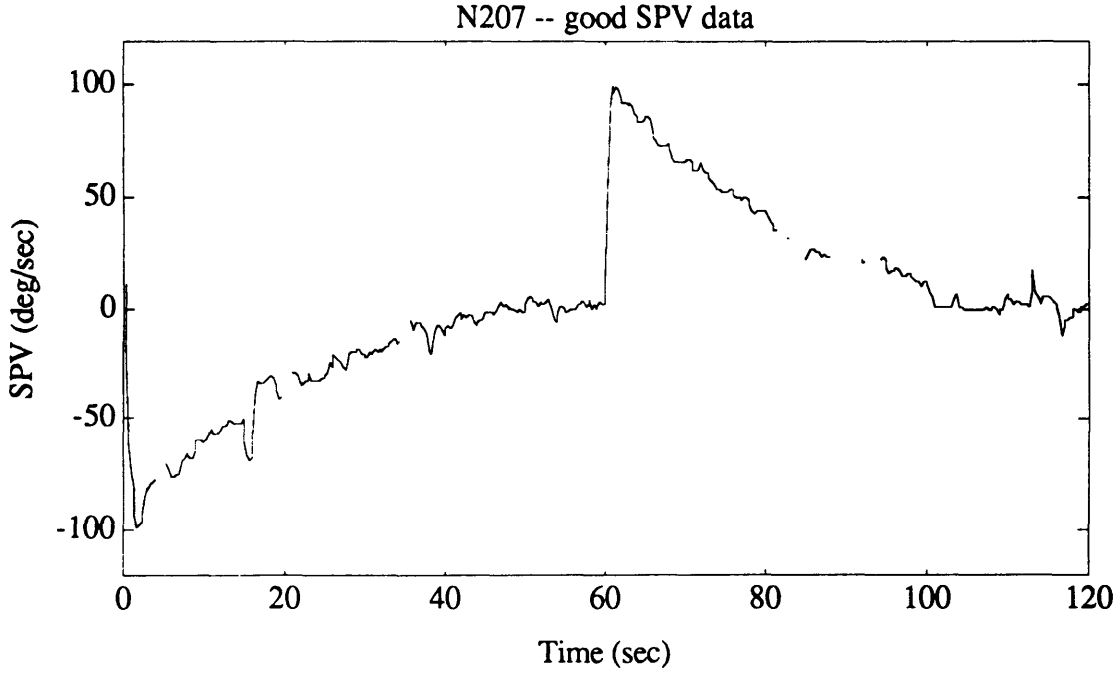


Figure 4.10d. Good portions of N207 SPV, after all outliers were removed.

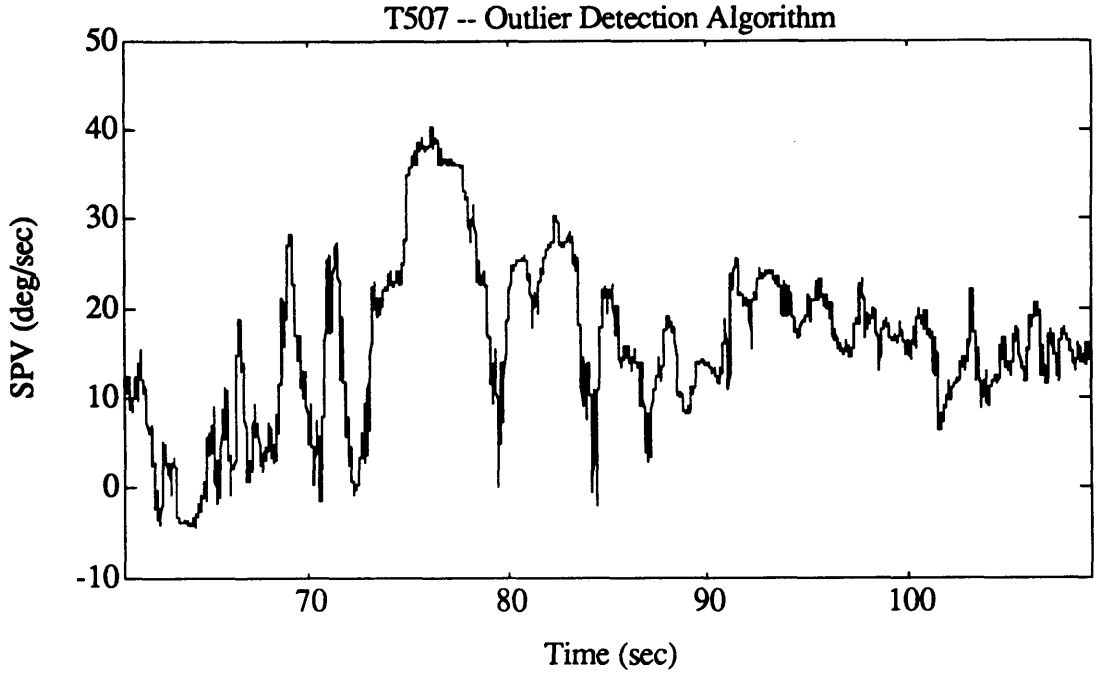


Figure 4.11a. Exponential portion of T507 post-rotatory SPV, as calculated by AATM.

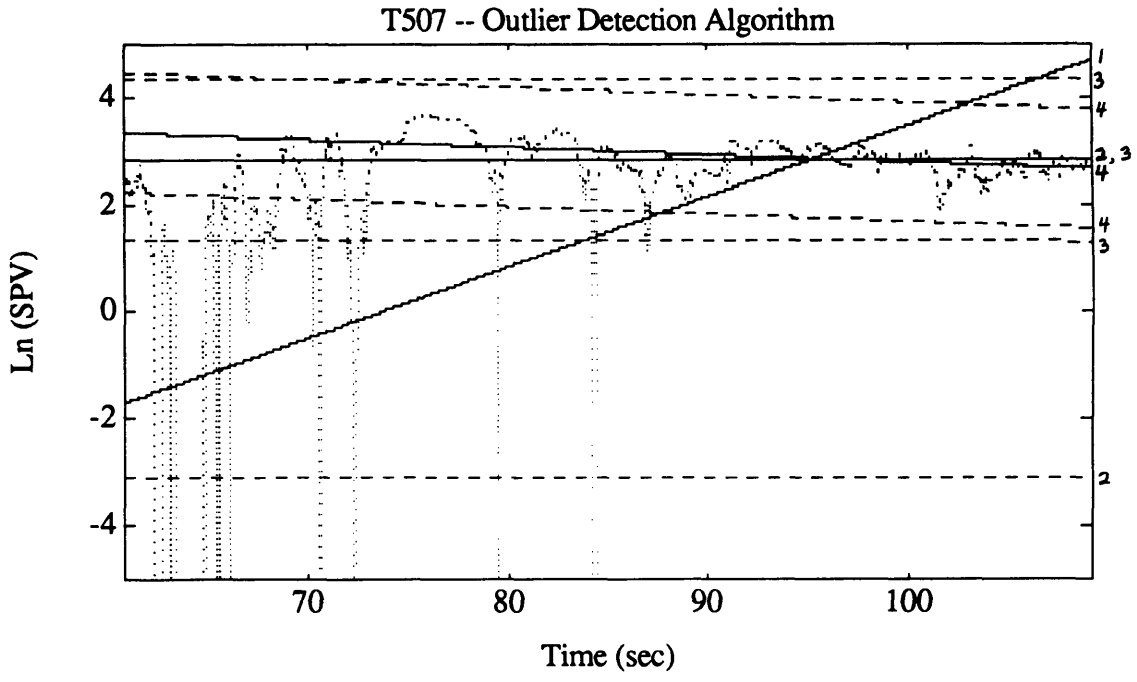


Figure 4.11b. Log-transformed data for exponential portion of T507 post-rotatory SPV, together with first four iterations of outlier detection algorithm. Dotted line is SPV data, solid lines are lines of best fit for each iteration, and dashed lines indicate bandwidths of ±3 RMS values.

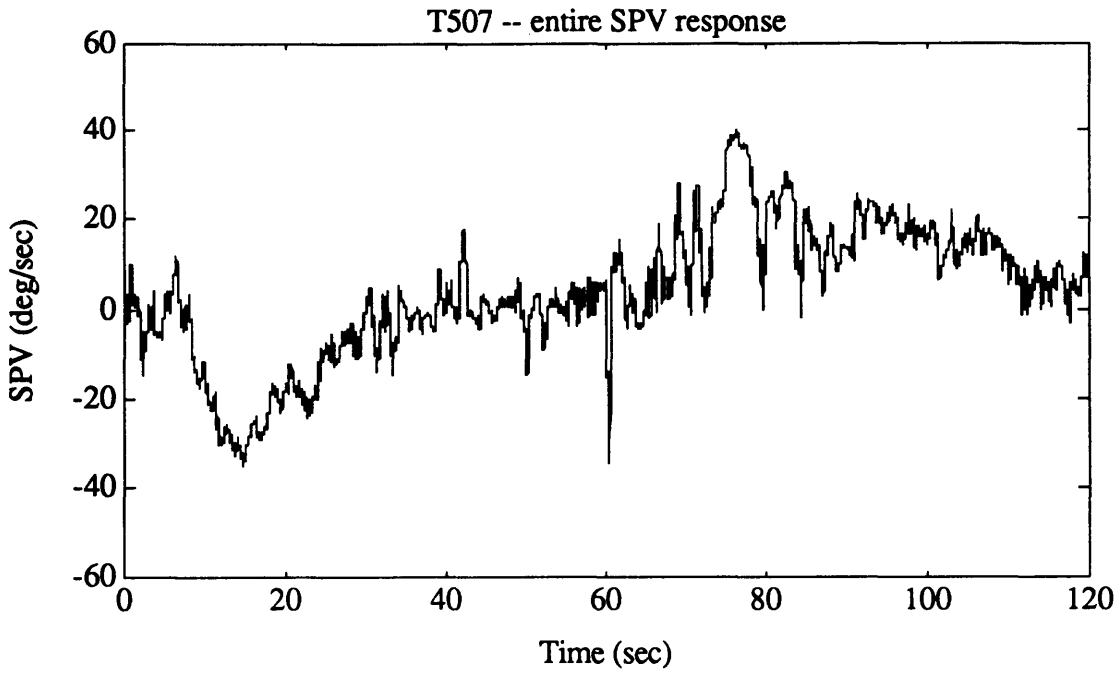


Figure 4.11c. Entire T507 SPV, as calculated by AATM.

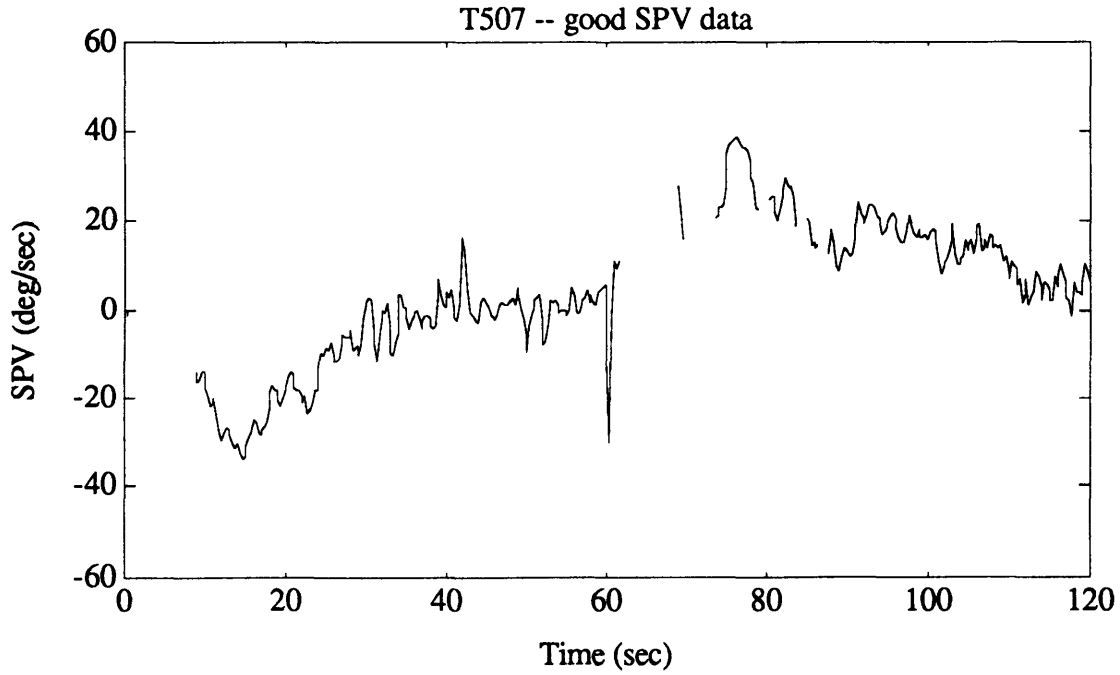


Figure 4.11d. Good portions of T507 SPV, after all outliers were removed.

4.6.3 Decimation

Now that the SPV data has been normalized for comparison, and the outliers indicated, it must be reduced to a manageable sampling rate. The data is subdivided into segments of thirty samples (0.25 s), and the mean and variance calculated for each segment. This results in a SPV time series which is resampled, or decimated, at 4 Hz but retains information about the variance within the original curve; this can be used for analysis of variance tests, although such tests were not performed as part of this thesis. Any outlier samples are excluded from these calculations; therefore, the mean and variance in a segment may be based upon fewer than thirty samples. If an entire segment is part of an outlier, that segment is marked in the decimated curve as an outlier.

4.7 Discarded Runs

The runs contained various amounts of data which were marked as bad by the outlier detection algorithm. In some cases, there were only a couple of short dropouts; these dropouts could be properly removed with the confidence that the rest of the trace was good. In other cases, a run would contain extensive dropouts with very little useful data, probably because the subject was not fully alert. These dropouts were fairly well detected by the automated algorithm.

In still more severe cases, there was virtually no response — the eye movements looked merely random and had no nystagmus evident in the eye position signal. The outlier detection algorithm was designed to remove the dropouts so that the remaining

data would reflect the real response of the subject. To be consistent with this goal, a run which exhibited little or no response should be discarded in its entirety.

It has been difficult to establish a single quantitative criterion for discarding an entire run. However, one can get a good idea as to the quality of the data by looking at the eye velocity signal obtained by differentiating the eye position. Figure 4.12 shows an example of the eye velocity for a high-quality run. The characteristic SPV envelopes for the per-rotatory and post-rotatory are evident. The fast phases are reflected in the larger magnitude velocity spikes in the opposite direction. The spacing of the spikes shows the beat frequency. Figure 4.13 shows an example of a run in which there was no nystagmus evident at any time in the position trace. Here, the eye velocity appears to be random noise, seldom exceeding 50 °/s. Figure 4.14 shows a run of intermediate quality. In the good portions, the velocity exhibits the character of Figure 4.12, while the dropout portions are shown by the relatively flat velocity sections characteristic of Figure 4.13. Such a run would be retained although extensive portions of its data would be automatically discarded by the outlier detection algorithm.

All of the SPV envelopes resulting from the *stat_prep* script were examined visually, and discarded if appropriate. Since the per- and post-rotatory sections may exhibit different data qualities, one half of the run may be discarded while the other half is retained. The runs which were discarded are tabulated in Section 5.3.

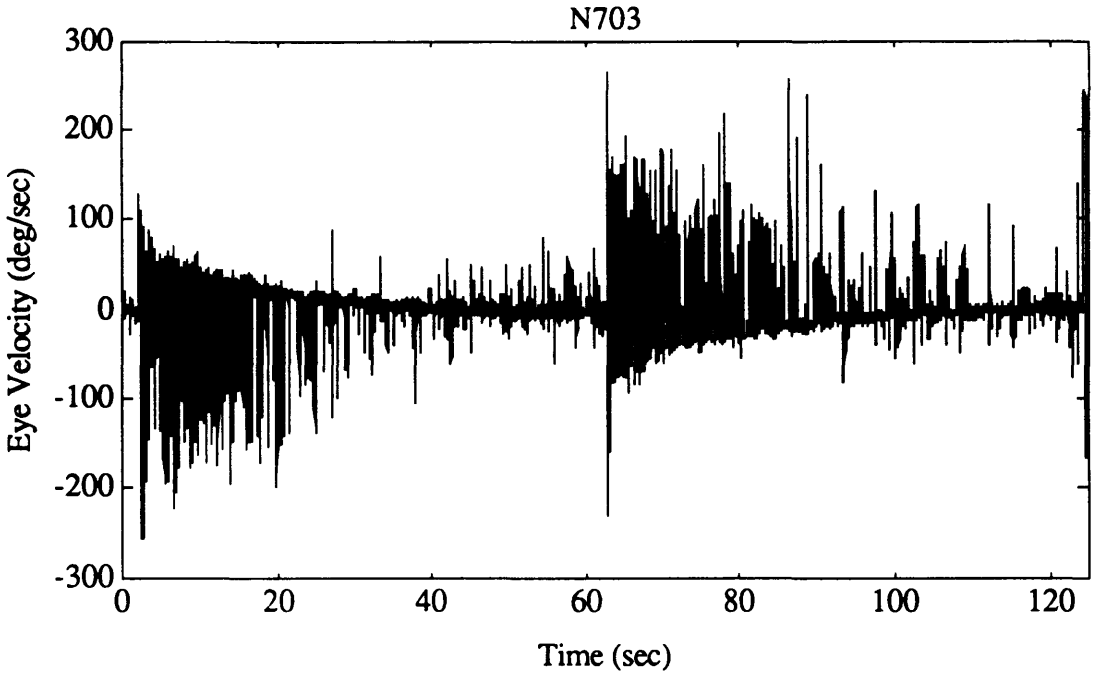


Figure 4.12. Eye velocity profile for a run exhibiting very high quality nystagmus (N703).

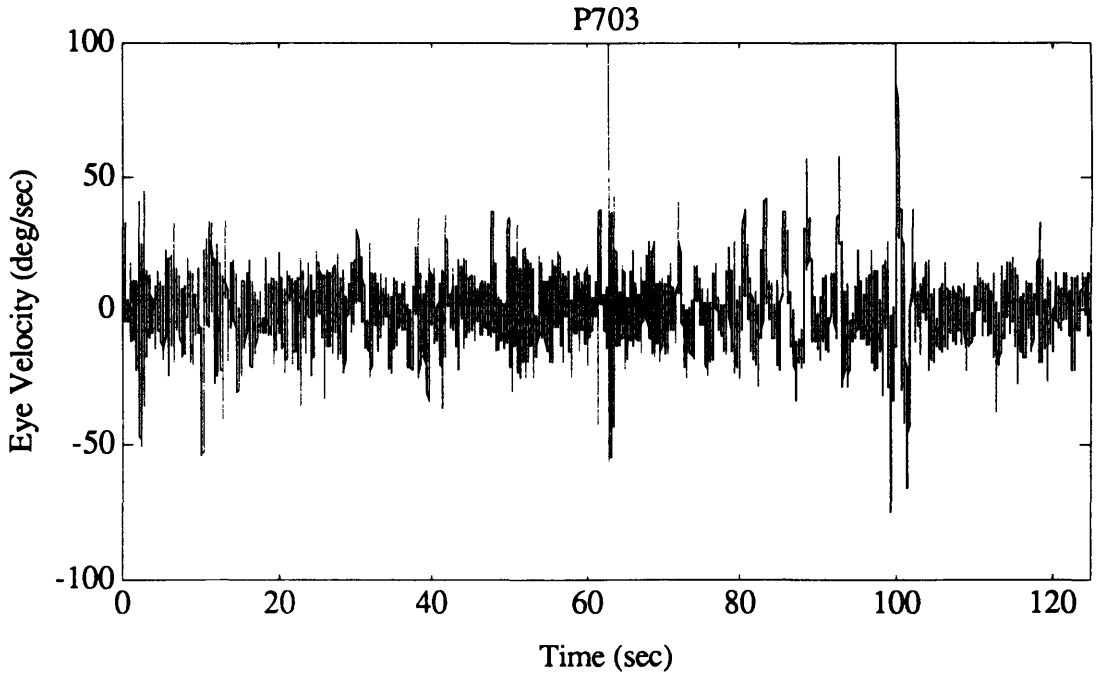


Figure 4.13. Eye velocity profile for a run exhibiting no nystagmus (P703).

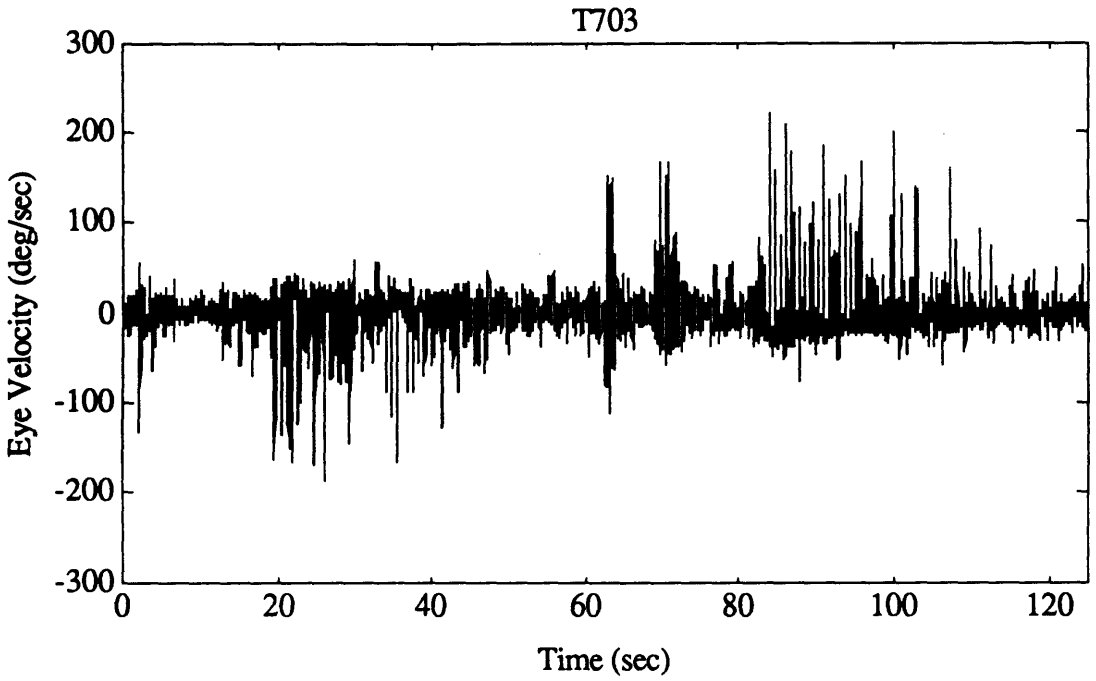


Figure 4.14. Eye velocity profile for a run exhibiting intermediate quality nystagmus (T703).

4.8 Comparison of SPV Profiles

4.8.1 Calculation of Test Statistic

For each subject, average SPV profiles were calculated by averaging the responses from individual decimated runs, yielding a mean $\bar{x}(t_i)$ and variance $s^2(t_i)$ at each point in time t_i. Since dropouts were not included in the calculations, the sample size $n(t_i)$ varied from point to point. This complicated the statistical comparison of the SPV profiles, as did the fact that $n(t_i)$ was small, occasionally zero. In the SL-1 and D-1 studies, the manual editing approach produced equal $n(t_i)$ at all points; however, this weighted the SPV data points unequally, with extra weight placed upon the points which determined the interpolations across the edited sections.

A five stage process was used to calculate a single test statistic for the comparison between two profiles. First, the variance for each of the two profiles was filtered by a five-point moving boxcar averager. If a variance were based on only two or three samples, it would be orders of magnitude smaller than neighbouring variances; this filtering took advantage of the fact that the variance of such physiological data is itself slowly varying over time. Second, those time points at which the mean SPV value for both traces and the sample variance were known, were determined. Third, a pooled variance estimate was calculated at each time point t_i by

$$s_p^2 = ((n_1 - 1)\, s_1^2 + (n_2 - 1)\, s_2^2) / (n_1 + n_2 - 2)$$

where the subscripts 1 and 2 refer to the first and second profiles respectively. Notice that it was assumed that the variances of the two profiles were equal. Fourth, a t-value

was determined at each point by a standard t-test as

$$t = \frac{\bar{x}_2 - \bar{x}_1}{\sqrt{s_p^2 (\frac{1}{n_1} + \frac{1}{n_2})}}$$

To explore one possible (if non-standard) measure of profile differences in a way that avoids the difficulties of the χ^2 parameter, we investigated the distribution of the test statistic $\lambda = \Sigma t^2$, summed over all points in time.

4.8.2 Sum of t-squares Distribution

In the SL-1 and D-1 studies, the values for λ were compared against a χ^2 distribution. This is only valid under the assumption that n_1 and n_2 are large; in reality, n_1 and n_2 are small so that λ should be compared against a sum of t-squared distribution (Σt^2) which depended upon the values of n_1 and n_2. Since these distributions were unknown, a Monte Carlo simulation technique was used to determine them (Pouliot, 1991). The simulation was done in MatLab by using normally distributed random values to determine λ values for profiles containing 100 points in time (100 degrees of freedom). A large number (5000) of these λ yielded a fairly good representation of the distribution, so that the percentiles could be confidently stated to within 5%. The *p*th percentile $\Lambda_{n_1,n_2,p}$ of the Σt^2 distribution corresponding to values n_1 and n_2 is defined as the value such that

$$Pr(\lambda \leq \Lambda_{n_1,n_2,p}) = p$$

The distribution of λ was determined for values of n_1 ranging from 2 to 20 in increments of 2, and for all integer values of n_2 ranging from 1 to 10. The percentiles

	n_2									
n_1	1	2	3	4	5	6	7	8	9	10
2	1000	504	245	186	160	145	137	131	127	123
4	244	185	159	146	136	131	126	124	121	118
6	161	146	137	131	126	124	121	118	117	115
8	137	131	126	124	120	119	117	115	114	112
10	126	123	121	119	117	115	115	113	112	111
12	121	118	117	115	114	113	112	111	111	110
14	117	115	114	114	112	112	111	110	110	109
16	114	113	112	112	111	111	109	109	109	108
18	113	111	111	110	109	110	109	108	108	107
20	111	110	110	109	109	109	108	107	108	107

Table 4.1. 0.5 percentiles for Σt^2 distribution for various values of n_1 and n_2.

	n_2									
n_1	1	2	3	4	5	6	7	8	9	10
2	1000	1000	546	318	242	209	189	177	169	163
4	537	306	240	203	187	176	169	163	158	155
6	243	207	188	179	169	163	158	155	152	151
8	190	177	168	164	158	155	151	150	148	144
10	170	164	158	154	152	151	147	145	144	143
12	158	154	151	149	148	145	144	142	141	141
14	151	150	148	146	144	144	141	140	140	140
16	147	145	144	143	142	141	139	140	138	138
18	145	143	142	142	140	139	138	138	137	137
20	142	140	139	140	138	138	137	136	136	136

Table 4.2. 0.95 percentiles for Σt^2 distribution for various values of n_1 and n_2.

	n_2									
n_1	1	2	3	4	5	6	7	8	9	10
2	1000	1000	680	374	265	227	201	190	179	172
4	702	354	268	219	202	189	179	172	166	164
6	272	226	201	192	179	173	167	163	160	159
8	202	188	178	173	166	163	160	157	155	152
10	180	173	167	162	160	158	155	152	152	150
12	166	162	160	157	156	152	152	149	148	148
14	159	158	155	153	152	150	148	146	146	147
16	156	152	151	149	148	148	145	146	145	144
18	151	150	150	148	146	146	145	144	144	143
20	149	147	146	146	145	144	144	143	142	141

Table 4.3. 0.975 percentiles for Σt^2 distribution for various values of n_1 and n_2.

	n_2									
n_1	1	2	3	4	5	6	7	8	9	10
2	1000	1000	968	492	311	251	222	207	191	184
4	1000	436	311	246	222	201	191	184	176	174
6	312	246	218	206	190	183	179	175	170	168
8	219	209	192	184	179	174	170	164	165	161
10	194	189	176	170	168	168	163	161	160	158
12	178	172	169	167	165	162	160	156	156	156
14	167	168	164	162	159	158	155	156	154	154
16	165	161	160	158	155	155	153	155	153	152
18	159	159	158	155	153	153	153	151	151	151
20	157	154	154	152	153	150	151	151	150	148

Table 4.4. 0.99 percentiles for Σt^2 distribution for various values of n_1 and n_2.

	n_2									
n_1	1	2	3	4	5	6	7	8	9	10
2	1000	1000	1000	1000	472	331	279	249	223	215
4	1000	807	526	307	295	244	226	207	195	204
6	525	348	271	263	223	214	207	199	203	187
8	288	253	227	211	205	197	197	189	178	181
10	241	222	207	199	191	190	188	182	183	178
12	210	191	193	193	182	180	188	178	177	173
14	190	196	183	183	181	175	174	173	173	169
16	187	182	180	173	174	176	169	167	170	169
18	174	175	174	177	173	170	168	166	171	166
20	176	173	176	174	172	168	169	168	171	166

Table 4.5. 0.999 percentiles for Σt^2 distribution for various values of n_1 and n_2.

for p = 0.5, 0.95, 0.975, 0.99, and 0.999 are shown in Tables 4.1 through 4.5 respectively. These are coarse values since they were generated by simulation. The critical values for p=0.999, for example, are coarser (because they are implicitly based on fewer simulation points) than those for lower p-values (say p=0.5). Since we desired a two-tailed test, the percentage point for standard statistical significance ($p < 0.05$) is given by Table 4.3.

These simulations were for 100 degrees of freedom. From smaller scale simulations, it appears that the ratio, *r*, of the Σt^2 percentiles to the number of degrees of freedom decreases slowly as the number of degrees of freedom increases. Pouliot (1991) simulated the distribution for (n_1=40, n_2=16) with 100 and 56 degrees of freedom; the ratio *r* for 5% significance changed from 1.29 to 1.37. As long as the number of degrees of freedom in a comparison is close to 100, the percentiles in the tables can be divided by 100 to yield significance levels for *r*. The number of degrees of freedom for

this project only varied from 90 to 156 (see Section 5.3); therefore, the tabulated values sufficed and were, in general, slightly conservative. The scripts to generate the distributions (*new_tsq*) and to calculate the percentiles (*tsq_pvalues*) are included in Appendix I.

These percentiles were calculated under the assumption that n_1 and n_2 were constant at all t_i. It would be extremely complicated to attempt to simulate the Σt^2 distributions, allowing either n_1 or n_2 to vary. For the data in this thesis, the values N_1 and N_2 were calculated as the mean n_1 and n_2 over all time points t_i; these N_1 and N_2 were used as the n_1 and n_2 in the tables to determine the appropriate percentile. Since the simulations were only performed for specific values of n_1 and n_2, the percentiles for a given comparison had to be interpolated from the tables. For simplicity, these interpolations were linear, although the tabulated values appeared to be non-linear. The result was that the interpolated values were in general slightly higher than they should have been. This would reject differences which were only marginally significant.

A difference was considered to be significant if the test statistic λ exceeded the percentile which was interpolated from the $p = 0.975$ table (Table 4.3). A trend towards significance was indicated if the test statistic λ exceeded the percentage point which was interpolated from the $p = 0.95$ table (Table 4.2).

4.9 Model Fitting

MatLab scripts were written to fit the model in Figure 2.5 to the data, and are included in Appendix J. The MatLab *constr* routine (Grace, 1990) performs a constrained optimization using a sequential quadratic programming (SQP) algorithm to minimize the RMS error between the actual data and the model data.

The per-rotatory and post-rotatory halves of the data were fit separately to reflect changes in parameters other than the gain, especially the "velocity storage" parameters g_0 and h_0 during dumping runs. Therefore, there was no need for an asymmetry parameter to reflect differences between per- and post-rotatory responses. During only one half of the profile, the SPV is unlikely to reach a sufficiently high magnitude in both CW and CCW directions to warrant a directional asymmetry parameter. Hence, the asymmetry parameter A was not included in the model fit, reducing the optimization to five parameters.

The model was fit to the section of data from 2 seconds after the start or stop (to allow transient chair motion, AATM response, and dumping head movements), to the full 60 seconds. The constraints initially placed on the model parameters were as follows:

$$0.2 \leq K \leq 1.2$$

$$2 \leq T_c \leq 15$$

$$30 \leq Ta \leq 300$$

$$10 \leq 1/h_0 \leq 60$$

$$0 \leq g_0 \leq 0.3$$

Unfortunately, the globally optimal model fit to the data frequently yielded results which were considered non-physiological (e.g. cupula time constants greater than 15

seconds, and velocity storage time constants less than the cupula time constant). This problem is typical of having too many free parameters. Therefore, the cupula and adaptation time constants were fixed at 6 and 80 seconds respectively. The ranges on the other three parameters were expanded to:

$$0.1 \leq K \leq 1.8$$

$$3 \leq 1/h_0 \leq 300$$

$$0 \leq g_0 \leq 0.45$$

These ranges were somewhat arbitrary but were intended to reflect the boundaries of what was considered "physiologically reasonable". The only parameter which ever reached the boundary of its constraint was g_0, which occasionally reached the ceiling of 0.45. A close inspection of these responses showed that, in most such cases, there was a delayed response at the beginning of the profile, followed by a slow increase in the SPV to its peak value. In the other cases, there was little or no data for the first 10 seconds. These situations resulted in tradeoffs between K and g_0 so that, as g_0 increased to 0.45, K decreased to values on the order of 0.15. Such a combination was considered to be physiologically unreasonable. All cases in which one of the model parameters reached a boundary were discarded.

Ideally, each individual run could be fit with this model to determine means and variances for the parameters. However, only the data from one subject (N) was of consistently high enough quality to be fit. The data from the other three subjects were averaged across sessions, and models were fit to these average responses. Model fits were also made to subject N's average responses and compared to the model fits to the individual runs.

5. Results

The rotating chair experiment for this work was successfully performed for the SLS-1 mission during four pre-flight sessions and five post-flight sessions in the week following the landing. The changes in the days of the pre-flight sessions from the originally intended dates were due primarily to delays in the launch of the space shuttle. An additional session was performed, about 430 days before the flight, which used a significantly different procedure. One extra session was also performed in-flight, but used significantly different equipment from the ground-based studies. Therefore, analysis of these additional sessions is not included here.

5.1 Completed Runs

Due to a number of various reasons, not all of the originally scheduled runs were performed in each session. 31 of the 352 runs were affected; no data was obtained for 14 of these runs and only partial data was obtained for the 17 others. The runs which were affected are summarized in Tables 5.1a through 5.1d.

The biggest loss was due to insufficient time in the half-hour session to perform all eleven runs in the protocol; this resulted in the incompletion of 8 stimulus runs in total. In two cases, the data collection computer malfunctioned in the middle of a run, so that data was only recorded for either the per-rotatory or post-rotatory phase but not both. One calibration run was lost due to operator error.

The remaining 20 runs were affected by medical complications. Due to earth sickness experienced by Subject P on the day of the landing, the second half of that session was

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
01	•	•	•	•	•	•	•	•
02	•	•	•	•	•	•	•	•
03	•	•	•	•	•	•	•	•
04	•	•	•	•	5	5	5	5
05	•	•	•	•	5	5	5	5
06	•	•	•	•	•	•	•	•
07	•	•	•	•	•	•	•	•
08	•	•	•	•	•	•	•	•
09	•	•	•	•	5	5	5	5
10	•	•	•	•	1	5	5	5
11	•	•	•	•	•	•	•	•

Table 5.1a. Completion status of runs which were performed for Subject M.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
01	•	•	•	•	•	•	•	•
02	•	•	•	•	•	•	•	•
03	•	•	•	•	•	•	•	•
04	•	•	•	•	•	•	•	•
05	3	•	•	•	•	•	•	•
06	•	•	•	•	•	•	•	•
07	•	•	•	•	•	•	•	•
08	•	•	•	•	•	•	•	•
09	•	•	•	•	•	1	•	•
10	•	•	•	•	1	1	1	•
11	•	•	•	•	•	•	•	•

Table 5.1b. Completion status of runs which were performed for Subject N.

Legend:
- • = performed normally
- 1 = insufficient time
- 2 = computer crash -- loss of per-rotatory data
- 3 = computer crash -- loss of post-rotatory data
- 4 = run not performed due to nausea
- 5 = no dumping maneuver due to unrelated injury
- 6 = loss of data due to operator error
- 7 = run performed in opposite direction due to error

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
01	•	•	•	•	•	•	•	6
02	•	•	•	•	•	•	•	•
03	7	•	•	•	•	•	•	•
04	•	•	•	•	•	•	•	•
05	•	•	•	•	•	•	•	•
06	•	•	•	•	•	•	•	•
07	•	•	•	•	4	•	•	•
08	2	•	•	•	4	•	•	•
09	1	•	•	•	4	•	•	•
10	1	•	•	•	4	•	•	•
11	•	•	•	•	4	•	•	•

Table 5.1c. Completion status of runs which were performed for Subject P.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
01	•	•	•	•	•	•	•	•
02	•	•	•	•	•	•	•	•
03	•	•	•	•	•	•	•	•
04	•	•	•	•	•	•	•	•
05	•	•	•	•	•	•	•	•
06	•	•	•	•	•	•	•	•
07	•	•	•	•	•	•	•	•
08	•	•	•	•	•	•	•	•
09	•	•	•	•	•	•	•	•
10	•	1	•	•	•	•	•	•
11	•	•	•	•	•	•	•	•

Table 5.1d. Status of runs which were performed for Subject T.

Legend:
- • = performed normally
- 1 = insufficient time
- 2 = computer crash -- loss of per-rotatory data
- 3 = computer crash -- loss of post-rotatory data
- 4 = run not performed due to nausea
- 5 = no dumping maneuver due to unrelated injury
- 6 = loss of data due to operator error
- 7 = run performed in opposite direction due to error

aborted, resulting in the loss of five runs (four stimulus, one calibration). Due to an unrelated back injury shortly after return to earth, Subject M was not able to perform the dumping manoeuvre; in lieu of this, normal head-upright runs were performed. Subject M also had to take pain relieving medication during the post-flight sessions.

5.2 Button Push Data

5.2.1 Raw Data

The raw values for the button push times, as calculated by a combination of the *tachan* script and direct data inspection, are tabulated in Tables 5.2a through 5.2h. The values shown are times in seconds from the start of either the acceleration (for T1) or deceleration (for T2) of the chair. A horizontal bar indicates a completely missing data point; either the run was not performed, the data was not recorded, or the subject forgot to press the button when the sensations disappeared.

In a large number of the runs, there was no button push for either or both of the per-rotatory and post-rotatory phases. An asterisk indicates that the sensation persisted for longer than the per-rotatory or post-rotatory phase of the motion, so that the button was not pushed. A value of 61 seconds is assigned to these data points, since this is longer than the chair rotation in most cases and allows us to include these data points in the statistical analyses. In these cases, the subjects generally reported that the sensations were at a very low level; therefore, 61 seconds is at least a reasonable value, and it is unlikely that a much greater value would have been obtained had chair rotation continued.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2	*	52.38	*	*	55.87	37.83	55.08	52.16
3	*	51.55	51.84	43.83	*	29.23	57.07	*
4	*	59.48	*	56.30	51.01	*	*	59.52
5	*	43.00	36.42	50.53	*	56.58	*	*
7	*	38.60	55.76	*	40.74	46.28	*	53.52
8	*	47.72	52.64	46.83	34.25	56.18	58.58	*
9	*	57.43	57.34	59.52	38.38	45.45	*	*
10	*	49.58	38.24	37.67	—	59.56	54.89	*

Table 5.2a. Per-rotatory button push times (T1) for Subject M.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2	60.81	44.52	47.77	42.09	35.08	16.38	24.03	57.47
3	*	44.34	41.08	40.29	30.21	17.14	47.09	52.64
4	11.04	14.84	23.75	10.31	*	33.27	51.73	48.69
5	10.15	13.28	23.00	10.62	61.53	36.38	60.42	45.32
7	*	40.17	45.45	38.71	32.20	46.67	42.96	43.58
8	44.38	44.61	*	45.95	31.60	37.89	42.89	47.33
9	15.21	23.99	20.67	16.02	39.81	41.15	52.07	45.89
10	16.48	17.55	20.73	12.17	—	41.03	*	48.08

Table 5.2b. Post-rotatory button push times (T2) for Subject M.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2	41.78	—	24.80	21.42	18.37	21.17	25.75	14.29
3	29.11	27.42	32.61	23.33	22.55	25.00	34.15	38.16
4	16.62	35.97	30.22	31.57	31.23	25.27	40.79	21.57
5	23.17	21.23	36.46	18.52	19.77	20.26	34.08	32.62
7	21.65	35.90	40.33	29.39	24.47	10.28	22.29	—
8	18.19	25.90	26.79	28.80	19.50	31.16	55.33	29.89
9	46.56	39.89	22.83	17.26	18.07	—	30.07	22.43
10	22.26	49.33	31.66	15.49	—	—	—	34.92

Table 5.2c. Per-rotatory button push times (T1) for Subject N.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2	29.20	34.51	33.42	18.03	20.11	21.32	15.31	18.14
3	44.90	27.14	28.73	27.81	19.36	26.14	27.33	27.29
4	55.22	11.64	22.32	24.66	—	9.78	17.20	9.30
5	—	18.54	22.12	22.17	17.38	12.69	13.89	15.80
7	20.93	23.89	22.67	30.38	24.68	17.02	16.73	34.42
8	30.57	32.72	27.45	33.84	27.04	29.60	31.62	28.82
9	45.02	34.78	10.82	13.82	8.93	—	19.59	14.54
10	17.51	28.97	11.75	14.09	—	—	—	18.07

Table 5.2d. Post-rotatory button push times (T2) for Subject N.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2	*	*	*	*	59.84	*	11.16	7.59
3	*	*	*	*	23.27	*	19.21	8.32
4	*	*	*	*	40.00	*	40.21	19.92
5	*	*	*	43.88	49.61	26.55	22.08	15.08
7	*	*	*	10.80	—	*	13.17	16.40
8	*	*	*	23.44	—	48.25	20.55	*
9	—	*	*	46.59	—	59.18	56.48	15.33
10	—	*	*	7.18	—	25.73	40.77	36.93

Table 5.2e. Per-rotatory button push times (T1) for Subject P.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2	16.45	25.58	34.77	*	*	*	32.52	33.88
3	19.95	27.08	29.93	*	22.26	*	26.58	34.48
4	15.54	7.03	12.36	—	7.81	15.84	16.80	7.93
5	18.41	8.43	9.49	12.63	13.82	8.75	8.03	7.98
7	45.42	27.53	21.55	23.44	—	19.88	29.47	34.17
8	27.88	18.87	16.24	24.03	—	19.68	28.00	23.31
9	—	11.17	5.28	14.57	—	9.35	9.44	9.67
10	—	13.97	7.26	10.22	—	7.48	4.70	11.82

Table 5.2f. Post-rotatory button push times (T2) for Subject P.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2	54.39	—	38.34	30.45	32.56	26.13	26.03	35.91
3	56.55	25.56	27.25	37.78	37.29	31.10	31.76	37.47
4	41.42	26.55	31.45	34.47	42.89	31.94	26.67	36.66
5	24.20	21.75	51.27	32.39	25.66	43.10	39.39	37.01
7	40.30	22.48	31.29	39.22	59.04	45.37	28.55	23.18
8	35.48	31.18	33.12	38.65	30.24	51.97	33.82	35.96
9	55.34	24.98	42.19	34.31	42.75	34.42	22.37	46.65
10	41.26	—	34.85	31.22	29.05	39.51	28.74	46.19

Table 5.2g. Per-rotatory button push times (T1) for Subject T.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2	29.55	25.57	18.65	29.95	34.16	17.76	24.10	30.63
3	24.67	20.32	17.36	31.25	22.68	19.23	21.17	20.94
4	15.74	8.91	6.04	6.39	2.73	2.83	5.47	1.56
5	23.73	6.73	6.12	2.50	2.52	2.18	2.36	2.17
7	23.41	21.98	18.61	18.75	18.51	20.35	16.36	19.31
8	21.57	19.33	24.12	22.41	16.21	19.48	17.88	21.57
9	14.01	5.47	6.88	3.62	2.71	1.62	2.05	2.07
10	12.53	—	4.81	3.92	5.83	3.41	1.67	2.72

Table 5.2h. Post-rotatory button push times (T2) for Subject T.

5.2.2 Pre-Flight Baseline

The pre-flight button push data was analyzed to determine baseline values for each subject under each condition. This yielded a profile of each subject's responses and gave the expected values and distributions for the post-flight button push times, given the null hypothesis that there was no change caused by exposure to weightlessness. Therefore, it was important to determine this pre-flight distribution with the highest possible confidence.

The data was first inspected across the pre-flight days to determine if there were any trends across sessions, or sessions which were different from the others. Some sessions appeared anomalous (see below). The data was averaged across those sessions which were consistent with each other. This yielded distributions for both T1 and T2 for each of the four stimulus types (CW head-up, CCW head-up, CW dump, and CCW dump).

Some of these eight distributions arguably should be equivalent. In particular, the per-rotatory stimulus for a CW head-up run was physically identical to that for a CW dump run. Therefore, T1 would have been expected to be the same under each condition; if it were, the values from the two conditions could be averaged, thereby increasing the number of data points and our confidence in the results. Similarly, if there were no directional asymmetry, the CW and CCW values could be averaged. T1 and T2 themselves could be averaged if there were no difference between per-rotatory and post-rotatory responses.

5.2.2.1 Anomalous Sessions

For subject M, the data from BDC2 was discarded. All eight runs yielded a T1 value greater than 61 seconds. From comments made by the subject during the session, she was experiencing difficulty in distinguishing between the vestibular cues and the cognitive knowledge that she was still rotating. Subject M had less prior training on the experiment than the others; with the practice from this session and subsequent training sessions, the subject seemed to have resolved this difficulty before engaging in the other three pre-flight sessions. In addition, the first three of the four head-up runs

yielded a T2 value greater than 60 seconds, whereas the fourth run yielded a T2 at the same level as the other sessions. While the post-rotatory sensations did not have the complications of the per-rotatory sensations (cognitive knowledge, wind cues, proprioceptive), it appears that there was a training effect here as well. Therefore, subject M's pre-flight averages were based upon only BDC3, BDC4, and BDC5.

For subject N, the first two of the three dumping runs resulted in T2 times that were substantially longer than the remaining pre-flight data. The head movement associated with the dump can be very confusing and disconcerting the first few times; it is quite possible that this was responsible for the elevated times. The third T2 value was at a reasonable level relative to the other pre-flight values. The variance of these BDC2 values (380.44) was significantly ($p < 0.05$) larger than the variance of the rest of the pre-flight sessions (57.38) as determined by an F-test. Therefore, the average pre-flight T2 values were determined from the other three sessions. The rest of the BDC2 data was retained since it appeared reasonably consistent with the other pre-flight sessions.

For subject P, the BDC5 button push data were erratic. In the other three pre-flight sessions, the per-rotatory sensations exceeded the rotation (T1 > 61 seconds) in all 22 runs. During BDC5, the first three of the eight runs were also at this level; however, the other five were substantially lower, and two of the runs (10.80 and 7.18 seconds) were unreasonably low values for any of the subjects. The reason for this discrepancy is unknown. The best estimate for the normal pre-flight T1 value was in excess of 61 seconds, with no information about the variance. In addition, the T2 values for the first two runs in BDC5 were in excess of 61 seconds, substantially longer than the previous sessions. Since this resulted in anomalous data in seven of the eight runs, it was considered most sound to discard the BDC5 data in its entirety.

For subject T, the T1 values appeared to be somewhat higher for BDC2 and somewhat lower for BDC3 than they were for BDC4 and BDC5. However, these deviations offset each other so that the average of BDC2 and BDC3 was consistent with the other sessions, and were probably only apparent due to the low number of trials. The inclusion of these two sessions in the pre-flight average response was to slightly increase the variance without affecting the mean. If one BDC were to be discarded, the other would also have to be, and this would have resulted in too few data points to make any significant judgments.

A separate issue for subject T was that the T2 values for the BDC2 dumping runs (mean = 16.50, variance = 24.93) were higher and more variable than for the other pre-flight sessions (mean = 5.58, variance = 3.19). As was the case for subject N, this was ascribed to training effects and the unfamiliar sensations associated with the dumping manoeuvre. Therefore, the dumping T2 averages were based upon the last three pre-flight sessions. The pattern of the 11 remaining pre-flight T2 times suggested a decreasing trend with subsequent exposure, due to either training or adaptation. This trend was indeed statistically significant as shown by a Spearman's rank correlation coefficient of $\rho_s = -0.718 > 0.623$ (Breiman, 1973).

Averaging the remaining data across all appropriate pre-flight sessions yielded the responses shown in Tables 5.3a through 5.3d. The mean, variance, and number of data points are shown for each stimulus condition.

	T1			T2		
Run	mean	var	n	mean	var	n
CW h-up	54.96	76.88	6	43.12	11.66	6
CCW h-up	49.07	12.19	6	46.21	57.24	6
CW dump	58.51	3.11	6	18.26	29.75	6
CCW dump	42.57	38.66	6	16.22	24.91	6

Table 5.3a. Average pre-flight button push times for subject M.

	T1			T2		
Run	mean	var	n	mean	var	n
CW h-up	30.75	74.54	7	26.63	36.93	8
CCW h-up	26.52	18.53	8	31.65	34.84	8
CW dump	30.11	114.70	8	19.68	87.47	6
CCW dump	27.26	126.28	8	19.61	38.75	6

Table 5.3b. Average pre-flight button push times for subject N.

	T1			T2		
Run	mean	var	n	mean	var	n
CW h-up	*	—	—	28.55	105.70	6
CCW h-up	*	—	—	23.33	32.00	6
CW dump	*	—	—	10.28	17.10	5
CCW dump	*	—	—	11.51	21.32	5

Table 5.3c. Average pre-flight button push times for subject P.

	T1			T2		
Run	mean	var	n	mean	var	n
CW h-up	36.64	100.94	7	23.31	22.07	8
CCW h-up	35.70	92.73	8	22.63	17.98	8
CW dump	36.34	96.78	8	6.22	3.01	6
CCW dump	33.85	101.30	7	4.82	2.88	5

Table 5.3d. Average pre-flight button push times for subject T.

5.2.2.2 Head-up T1 vs. Dumping T1

Since the spin phase of the stimulus to the vestibular system was the same for both head-up and dumping runs, it was expected that the responses would be the same. The average pre-flight T1 button push times were tested for each subject by comparing CW head-up runs against CW dumping runs, and CCW head-up runs against CCW dumping runs. An F-test was used to check for equal variance, and a t-test was used to check for a difference in the mean. Table 5.4 shows the F ratio, t value, pooled variance, and p-values for each direction for each subject. It was assumed that the button push data was normally distributed, and that the effects of the 61 second maximum would be minimal.

Subj	Dir	F	df	p	pooled	df	t	p
M	CW	24.72	(5,5)	< .01	40.00	10	-.972	> .3
	CCW	3.17	(5,5)	> .2	25.43	10	2.233	< .05
N	CW	1.54	(7,6)	> .2	96.16	13	.126	> .5
	CCW	6.81	(7,7)	< .05	72.41	14	-.174	> .5
T	CW	1.04	(6,7)	> .2	98.70	13	.058	> .5
	CCW	1.09	(6,7)	> .2	96.69	13	.364	> .5

Table 5.4. Summary of F-test and t-test results performed on pre-flight data. Comparisons were made between T1 values from head-up runs and T1 values from dumping runs. The first two columns show the subject letter and direction of the spin phase of the stimulus. The F-ratio is shown for each case, together with the number of degrees of freedom for the numerator and denominator respectively, and the associated p-value. The pooled variance estimate, number of degrees of freedom, t-value, and p-value are shown for a standard t-test. A negative sign in the t-value indicates that the mean from the head-up runs was larger than that from the dumping runs.

The variability of the CW head-up T1 values for subject M was significantly higher than the variability in the corresponding dumping runs; however, this could be attributed to an uncharacteristically high head-up variance (due to one low time of 38.6

seconds), and an uncharacteristically low dumping variance. Meanwhile, the variability in the CCW head-up T1 values was somewhat, but not significantly, lower than the variability in the dumping runs. The difference between the mean CCW T1 values was barely significant; however, the sign of the difference was inconsistent with the slight difference in the CW runs. It appeared therefore that any differences were simply due to the low number of data points; hence, the head-up and dumping T1 values were averaged for each direction for subject M.

The data for subjects N and T showed no difference in the mean T1 values. The variability in the CCW head-up T1 values was uncharacteristically low, resulting in a significant F ratio; however, the mean T1 values were equivalent in spite of this. Therefore, the head-up and dumping T1 values were averaged for both subjects.

It was impossible to perform similar tests on the data for subject P, for reasons described earlier. The results of averaging the head-up and dumping T1 data together are shown in Table 5.5.

		T1		
Subject	Direction	mean	variance	n
M	CW	56.73	39.81	12
	CCW	45.82	34.61	12
N	CW	30.41	89.39	15
	CCW	26.89	67.74	16
T	CW	36.48	91.73	15
	CCW	34.83	90.70	15

Table 5.5. Summary of pre-flight T1 data obtained by averaging head-up and dumping runs for each subject and direction. The mean, variance and number of data points are shown. Notice that the variance shown is calculated by treating head-up and dumping runs as being from the same population, and therefore differs slightly from the pooled variance estimate in Table 5.4.

5.2.2.3 Directional Asymmetry

The pre-flight values were similarly analyzed for directional asymmetries in the responses of the three subjects M, N, and T. Comparisons were made between CW T1 values and CCW T1 values, between CW head-up T2 values and CCW head-up T2 values, and between CW dumping T2 values and CCW dumping T2 values. F-tests were used to check for equal variances, and t-tests for equal means. The resultant test data are shown in Table 5.6.

Subj	Time	F	df	p	pooled	df	t	p
M	T1	1.15	(11,11)	> .2	37.21	22	4.381	< .001
	h-up T2	4.91	(5,5)	> .1	34.45	10	-.912	> .3
	dump T2	1.19	(5,5)	> .2	27.33	10	.676	> .5
N	T1	1.32	(14,15)	> .2	78.19	29	1.108	> .2
	h-up T2	1.06	(7,7)	> .2	35.89	14	-1.676	> .1
	dump T2	2.26	(5,5)	> .2	63.11	10	.015	> .5
P	T1	—	—	—	—	—	—	—
	h-up T2	3.30	(5,5)	> .2	68.85	10	1.089	> .3
	dump T2	1.25	(4,4)	> .2	19.21	8	-.444	> .5
T	T1	1.01	(14,14)	> .2	91.22	28	.473	> .5
	h-up T2	1.23	(7,7)	> .2	20.03	14	.304	> .5
	dump T2	1.05	(5,4)	> .2	2.95	9	1.347	> .2

Table 5.6. Summary of F-test and t-test results performed on pre-flight data for directional asymmetry. Comparisons were made between T1 values from CW and CCW runs, between T2 values from head-up CW and CCW runs, and between T2 values from dumping CW and CCW runs. The first two columns show the subject letter and which T1 or T2 values were being compared. The remaining columns are similar to those in Table 5.4. A negative sign in the t-value indicates that the mean from the CCW runs was larger than that from the CW runs.

In no cases were the variances significantly different. There was only a significant difference in the mean in one case, that of T1 values for subject M; here, the CW times were significantly larger than the CCW times. Therefore, the directions were not averaged for subject M. While the difference in the T2 head-up values was not significant, it should be noted that the sign of the difference was consistent with the T1 values since the direction of the post-rotatory portion of the stimulus to the vestibular system was opposite to that of the per-rotatory portion. The slight difference in the dumping T2 values was of the opposite sign, but the stimulus here was complicated by the confusing effects of the head movement.

For subject N, the differences in the times were not significant, but they were consistent. The T1 values from CW head-up rotations were 4 seconds longer than from CCW head-up rotations; the T1 values from CW dumping rotations were 3 seconds longer than from CCW dumping rotations; the T2 values from CCW head-up rotations (hence equivalent to a CW stimulus to the vestibular system) were 5 seconds longer than from CW head-up rotations (equivalent to a CCW stimulus). As such, a paired t-test was performed on each of the three sets of data, but none of the paired differences were significant (T1: $t = 1.304$, $df = 14$; h-up T2: $t = -1.825$, $df = 7$; dump T2: $t = 0.039$, $df = 5$). A paired t-test on the entire data set yielded a trend towards significance ($t = 1.965$, $df = 29$, $p < 0.1$). Since the differences were not significant, the directions were averaged together, but the differences were noted.

The button push data was averaged across directions for subjects N, P, and T. The summary of the pre-flight data so far is shown in Table 5.7.

Subject	Time	mean	variance	n
M	CW T1	56.73	39.81	12
	CCW T1	45.82	34.61	12
	CW h-up T2	43.12	11.66	6
	CCW h-up T2	46.21	57.24	6
	CW dump T2	18.26	29.75	6
	CCW dump T2	16.22	24.91	6
N	T1	28.59	78.79	31
	h-up T2	29.14	40.21	16
	dump T2	19.64	57.37	12
P	T1	*	-	-
	h-up T2	25.94	70.04	12
	dump T2	10.89	17.50	10
T	T1	35.66	88.74	30
	h-up T2	22.97	18.81	16
	dump T2	5.58	3.19	11

Table 5.7. Average pre-flight button push times, together with variance and number of data points. The second column shows the range over which the times have been averaged. For example, "CW dump T2" indicates that only the T2 times from the CW dumping runs were combined, whereas "T1" indicates that T1 times were averaged across both directions and both head movement types.

5.2.2.4 Per-rotatory versus Post-rotatory

The per-rotatory button push times (T1) were compared against the post-rotatory button push times (T2) to indicate the degree to which the extra cues during the rotation (i.e. wind, audio, and proprioceptive cues associated with chair vibration during rotation) were affecting the data. These factors would tend to lengthen T1. However, it should be noted that a reversal in the VOR (as would be expected at about 40 seconds) would tend to lengthen T2. This required comparisons between T1 times and head-up T2 times. In the case of a directional asymmetry, the CW T1 times were compared to the

CCW T2 times and the CCW T1 times were compared to the CW T2 times. As was noted earlier the post-rotary stimulus to the vestibular system was the mirror image of the per-rotatory stimulus, and hence the direction was reversed. The data from the F-tests and t-tests are shown in Table 5.8.

Subject	F	df	p	pooled	df	t	p
M (CW)	1.44	(5,11)	> .2	48.53	16	3.020	< .01
M (CCW)	2.97	(11,5)	> .2	23.14	16	1.123	> .2
N	1.96	(30,15)	> .1	59.50	45	-.232	> .5
T	4.72	(29,15)	< .01	53.78	44	5.590	< .001

Table 5.8. Summary of F-test and t-test results performed on pre-flight data for per-rotatory versus post-rotatory differences. The first column shows the subject letter and, if applicable, the direction of the T1 data being compared. The remaining columns are similar to those in Table 5.4. A negative sign in the t-value indicates that the mean T2 time was larger than the mean T1 time.

For subject M, the T1 times were significantly longer than the T2 times for the CW direction. The CCW direction showed similar behaviour although the difference was not significant.

For subject N, there was no significant difference between the mean T1 and T2 times, although the variability of the T2 times was half that of the T1 times.

For subject T, the variability of the T1 times was significantly greater than that of the T2 times. However, mean T1 was significantly longer than T2 ($p < .001$) level even if the pooled variance was taken to be the maximum of the two variances (88.74).

Thus, for two of the three subjects, the per-rotatory button push times were longer than the post-rotatory button push times, and were also more variable. Therefore, the distinction between T1 and T2 were maintained for all subjects (including N).

5.2.2.5 Dumping Effects

The T2 times from the head-up runs were compared with the T2 times from the dumping runs to determine the extent to which the sensations were affected by the head movement. All four subjects were analyzed, with both directions being analyzed separately for subject M. A quick glance demonstrated that the dumping runs produced much shorter T2 times than the head-up runs. Since the times were shorter, it was understandable that the standard deviation would correspondingly be lower since their ratio seemed to be slowly varying. A t-test was performed on the means, conservatively taking the variance to be the maximum of the variances of the head-up and dumping T2 times. The data from these tests is shown in Table 5.9. The values shown for *M (CW)* are the T2 times for the CW runs, despite the fact that the actual post-rotatory stimulus to the vestibular system was in the CCW direction. Similarly, the values shown for *M (CCW)* are the T2 times for the CCW runs, despite the fact that the actual post-rotatory stimulus to the vestibular system was in the CW direction.

The data clearly shows that the dumping manoeuvre significantly shortened the subjective responses, despite the fact that the variability of the dumping T2 times was significantly reduced from that of the head-up T2 times in two of the five cases.

Subject	F	df	p	var	df	t	p
M (CW)	2.55	(5,5)	> .2	29.75	10	7.894	< .001
M (CCW)	2.30	(5,5)	> .2	57.24	10	6.866	< .001
N	1.43	(11,15)	> .2	57.37	26	3.284	< .01
P	4.00	(11,9)	< .05	70.04	20	4.200	< .001
T	5.90	(15,10)	< .01	18.81	25	10.237	< .001

Table 5.9. Summary of F-test and t-test results performed on pre-flight data for effects of the head movement on the post-rotatory button push times. The first column shows the subject letter and, if applicable, the per-rotatory direction of the data being compared. The remaining columns are similar to those in Table 5.4. A negative sign in the t-value indicates that the mean dumping T2 time was larger than the mean head-up T2 time.

5.2.3 Post-flight Changes

Each subject was tested twice within 60 hours of the return to earth. The responses over these two sessions were averaged together and are referred to as the "return" data. Each subject was then tested 4 and 7 days after the landing to determine if the response had recovered to the pre-flight baseline. The responses from these two sessions were averaged together and are referred to as the "recovery" data. The average post-flight data is tabulated in Tables 5.10a and 5.10b in the same format as that of the pre-flight data in Table 5.7. Due to an unrelated back injury shortly after landing, subject M was unable to perform the dumping head movement; the manoeuvre was skipped, resulting in only head-up runs throughout the post-flight period.

Subject	Time	mean	variance	n
M	CW T1	47.07	70.17	8
	CCW T1	51.11	180.98	7
	CW h-up T2	38.19	164.20	8
	CCW h-up T2	36.54	181.25	7
	CW dump T2	—	—	—
	CCW dump T2	—	—	—
N	T1	22.08	31.34	13
	h-up T2	23.16	18.93	8
	dump T2	12.20	14.51	4
P	T1	48.04	234.18	12
	h-up T2	40.80	490.34	6
	dump T2	10.51	12.06	6
T	T1	37.69	88.40	16
	h-up T2	21.05	31.64	8
	dump T2	3.10	1.84	8

Table 5.10a. Average post-flight button push times for the "return" sessions, together with variance and number of data points. The second column shows the range over which the times have been averaged. For example, "CW dump T2" indicates that only the T2 times from the CW dumping runs were combined, whereas "T1" indicates that T1 times were averaged across both directions and both head movement types.

Subject	Time	mean	variance	n
M	CW T1	58.04	14.44	8
	CCW T1	59.44	5.61	8
	CW h-up T2	45.80	100.80	8
	CCW h-up T2	50.60	46.48	8
	CW dump T2	—	—	—
	CCW dump T2	—	—	—
N	T1	31.53	108.91	14
	h-up T2	24.96	52.47	8
	dump T2	15.49	11.38	7
P	T1	25.26	278.34	16
	h-up T2	30.30	16.97	8
	dump T2	9.55	12.66	8
T	T1	33.52	53.46	16
	h-up T2	21.49	19.28	8
	dump T2	2.51	1.56	8

Table 5.10b. Average post-flight button push times for the "recovery" sessions, together with variance and number of data points. The second column shows the range over which the times have been averaged. For example, "CW dump T2" indicates that only the T2 times from the CW dumping runs were combined, whereas "T1" indicates that T1 times were averaged across both directions and both head movement types.

5.2.3.1 T1 Changes

The return and recovery data was compared to the pre-flight data by F-tests for equal variances, and t-tests for differences in means. The test results are shown in Tables 5.11a and 5.11b for the return and recovery data respectively. If the variances of the two data sets were significantly different, the variance for the t-test was taken to be the larger of the two variances; otherwise, a pooled variance estimate was used. Figures 5.1a through 5.1e show graphically the raw data values, together with the mean (horizontal dashed line) and standard deviation (vertical dashed bars) of the pre-flight data.

Subject	F	df	p	var	df	t	p
M (CW)	1.76	(7,11)	> .2	51.62	18	2.946	< .05
M (CCW)	5.23	(6,11)	< .02	180.98	17	-.827	> .4
N	2.51	(30,12)	> .1	65.23	42	2.439	< .02
T	1.00	(29,15)	> .2	88.62	44	-.697	> .4

Table 5.11a. Summary of F-test and t-test results performed on post-flight (return) data for changes in the per-rotatory button push times. The columns shown are the same as in Table 5.6. A negative sign in the t-value indicates that the mean return T1 time was larger than the mean pre-flight T1 time.

Subject	F	df	p	var	df	t	p
M (CW)	2.76	(11,7)	> .1	29.94	18	-.524	> .5
M (CCW)	6.17	(11,7)	< .05	34.61	18	-5.072	< .001
N	1.38	(13,30)	> .2	87.90	43	-.974	> .3
T	1.66	(29,15)	> .2	76.71	44	.789	> .4

Table 5.11b. Summary of F-test and t-test results performed on post-flight (recovery) data for changes in the per-rotatory button push times. The columns shown are the same as in Table 5.6. A negative sign in the t-value indicates that the mean recovery T1 time was larger than the mean pre-flight T1 time.

For subject M, the directional asymmetry from the pre-flight data had apparently vanished. The differences between the CW and CCW directions were insignificant in both the return and recovery sessions. The variances were larger in the return data, and smaller in the recovery data, than in the pre-flight control data; the latter can be attributed to the fact that 9 out of the 16 recovery T1 values were truncated at 61 seconds, thereby causing a compression of the range of the data.

The mean CW T1 time was significantly lower initially after the return to earth, but had recovered to the pre-flight levels by the end of the week. The mean CCW time however was initially unchanged, but reached a significantly higher level by the end of the week. This difference was apparently due to the lower pre-flight CCW T1 values.

Figure 5.1a shows that 7 of the 8 CW return runs yielded times lower than the pre-flight mean (referred to henceforth as below average). Similarly, all 8 of the CCW recovery runs yielded times greater than the pre-flight mean (referred to henceforth as above average). This type of non-parametric sign test (Rosner, 1986) is useful for confirmation.

For subject N, the T1 time was significantly lower during the return phase, as reflected by the fact that 11 out of the 13 runs were below average (sign test, $p < 0.0225$). The T1 time had recovered to its pre-flight level by the end of the week; the individual data points were split with 5 below average and 9 above average.

For subject P, there was so much variability in the post-flight data that no conclusions could be drawn. The T1 times were as erratic post-flight as they were on the last pre-flight session, suggesting that this variability may be due to a change in the subject's reporting criteria which occurred between BDC4 and BDC5.

For subject T, the post-flight T1 times apparently did not change from the pre-flight levels; the variances and mean values were equivalent to the pre-flight levels during both the return and recovery phases. Even the individual data points were equally split with 8 above and 8 below average.

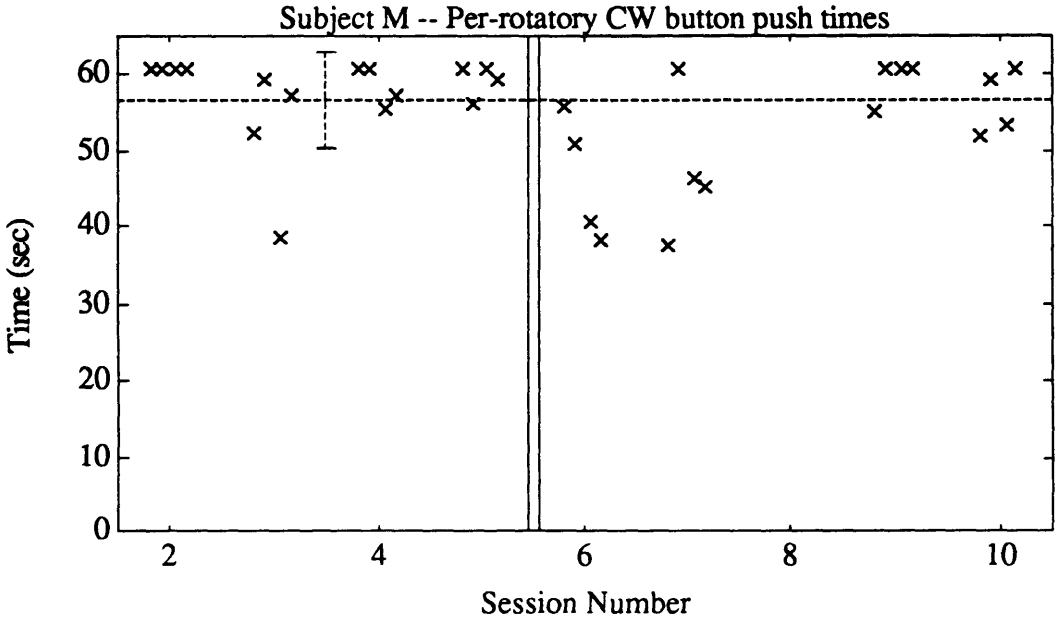


Figure 5.1a. Clockwise per-rotatory button push times for Subject M. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

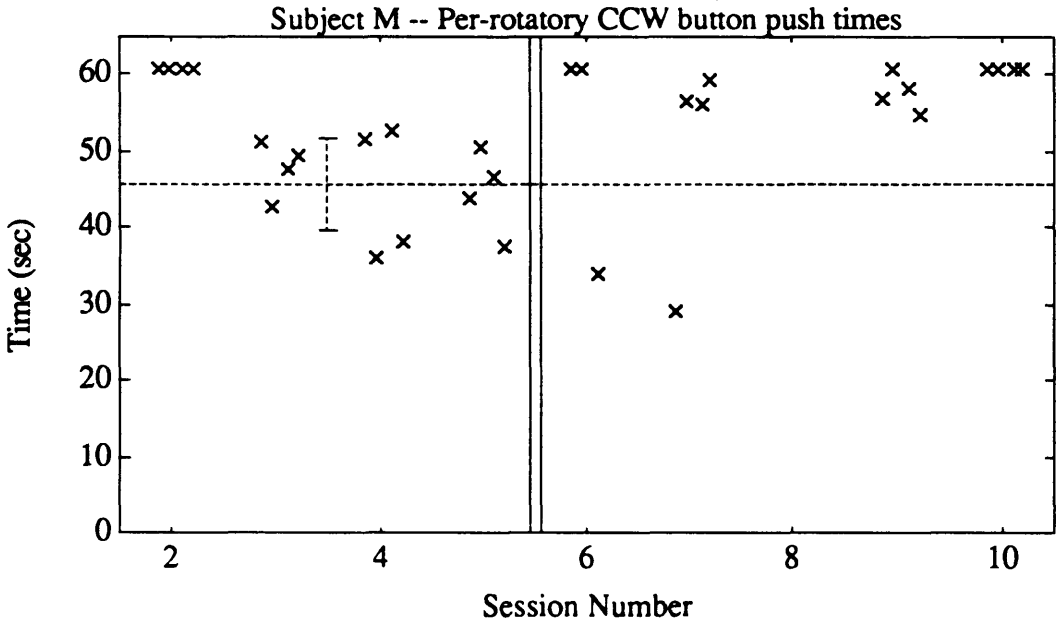


Figure 5.1b. Counter-clockwise per-rotatory button push times for Subject M. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

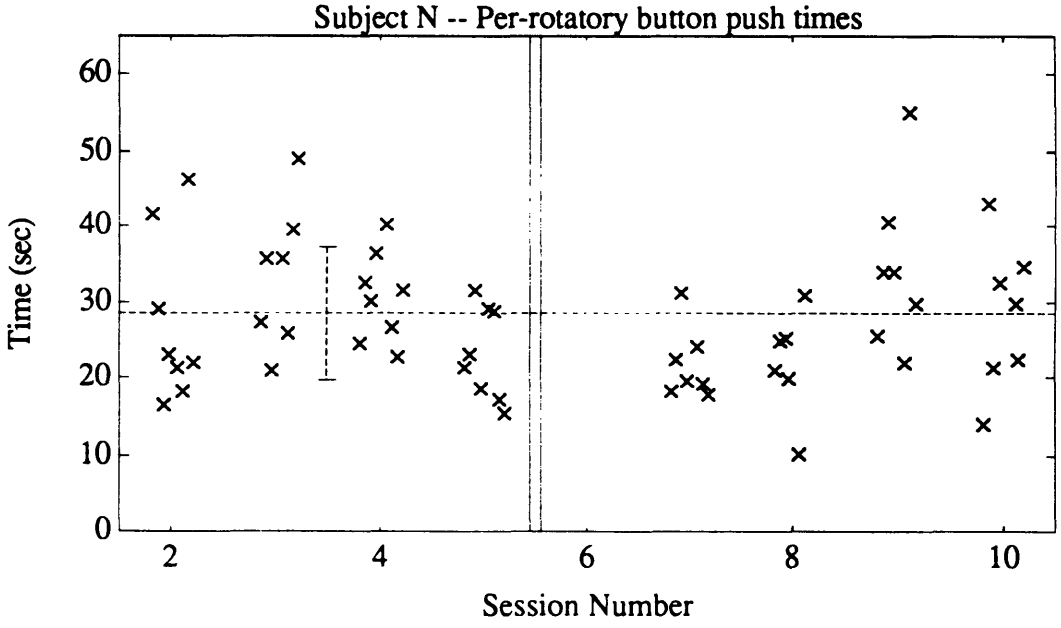


Figure 5.1c. Per-rotatory button push times for Subject N. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

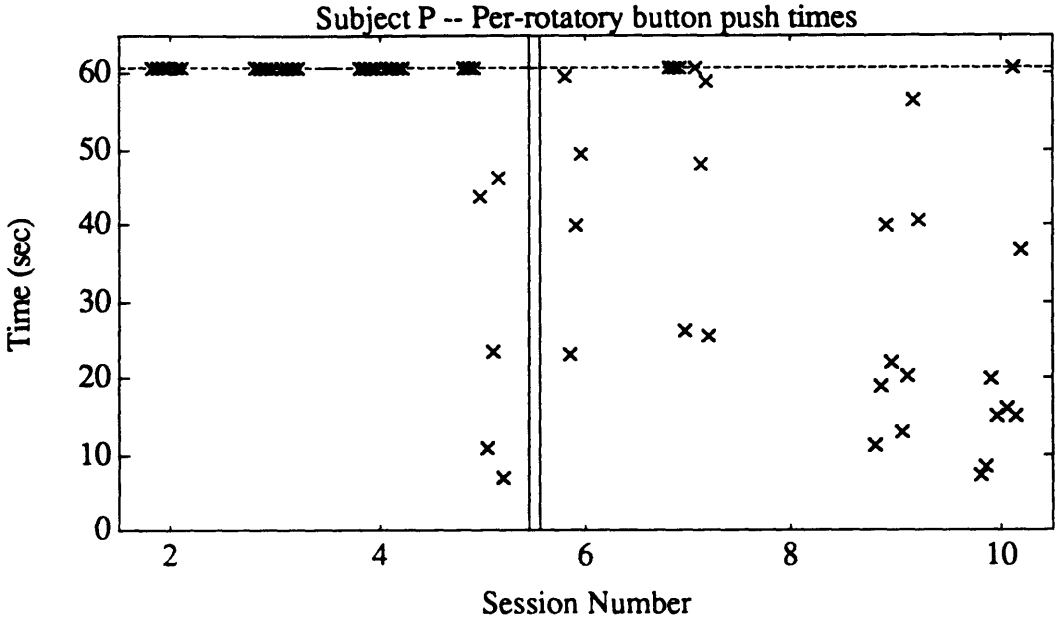


Figure 5.1d. Per-rotatory button push times for Subject P. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

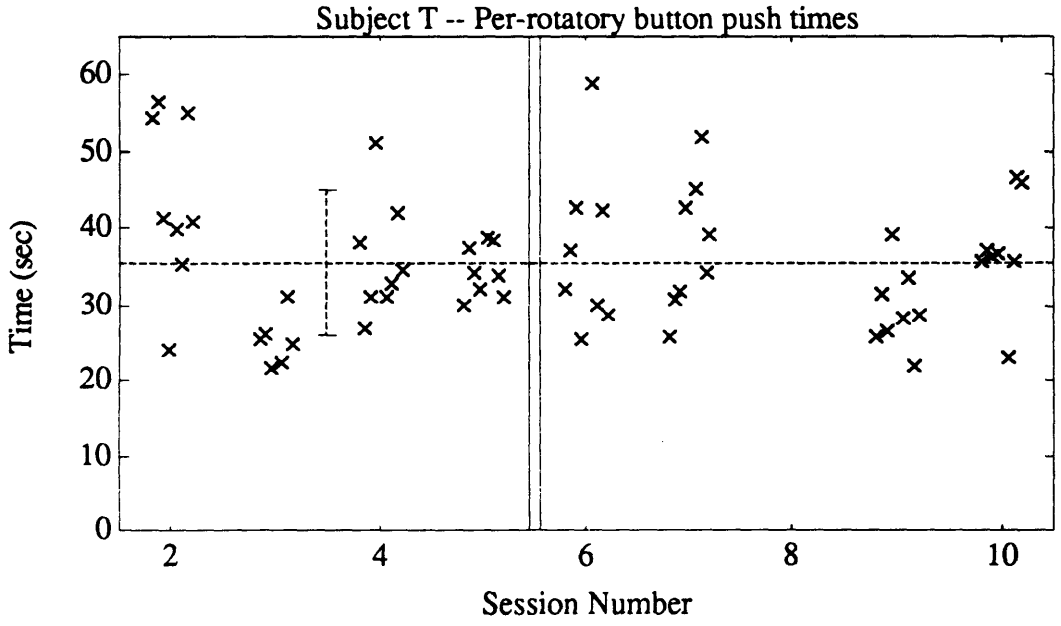


Figure 5.1e. Per-rotatory button push times for Subject T. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

5.2.3.2 Head-up T2 Changes

The return and recovery data was compared to the pre-flight data by F-tests for equal variances, and t-tests for differences in means. The test results are shown in Tables 5.12a and 5.12b for the return and recovery data respectively. If the variances of the two data sets were significantly different, the variance for the t-test was taken to be the larger of the two variances; otherwise, a pooled variance estimate was used. Figures 5.2a through 5.2e show graphically the raw data values, together with the mean (horizontal dashed line) and standard deviation (vertical dashed bars) of the pre-flight data.

Subject	F	df	p	var	df	t	p
M (CW)	14.08	(7,5)	< .02	164.20	12	.712	> .4
M (CCW)	3.17	(6,5)	> .2	124.88	11	1.556	> .1
N	2.12	(15,7)	> .2	33.44	22	2.388	< .05
P	—	—	—	—	—	—	—
T	1.68	(7,15)	> .2	22.89	22	.927	> .3

Table 5.12a. Summary of F-test and t-test results performed on post-flight (return) data for changes in the post-rotatory button push times for head-up runs. The columns shown are the same as in Table 5.6. A negative sign in the t-value indicates that the mean return head-up T2 time was larger than the mean pre-flight time.

Subject	F	df	p	var	df	t	p
M (CW)	8.64	(7,5)	< .05	100.80	12	-.695	> .4
M (CCW)	1.23	(5,7)	> .2	50.96	12	-1.139	> .2
N	1.30	(7,15)	> .2	44.11	22	1.453	> .1
P	4.13	(11,7)	< .1	70.04	18	-1.141	> .2
T	1.02	(7,15)	> .2	18.96	22	.785	> .4

Table 5.12b. Summary of F-test and t-test results performed on post-flight (recovery) data for changes in the post-rotatory button push times for head-up runs. The columns shown are the same as in Table 5.6. A negative sign in the t-value indicates that the mean recovery head-up T2 time was larger than the mean pre-flight time.

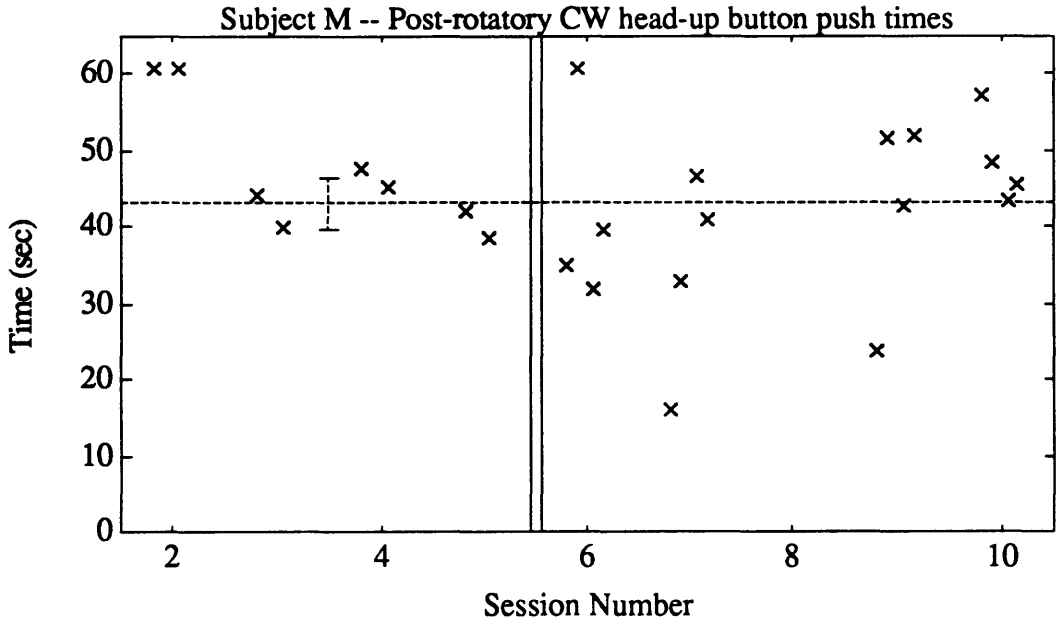


Figure 5.2a. Clockwise head-up post-rotatory button push times for Subject M. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

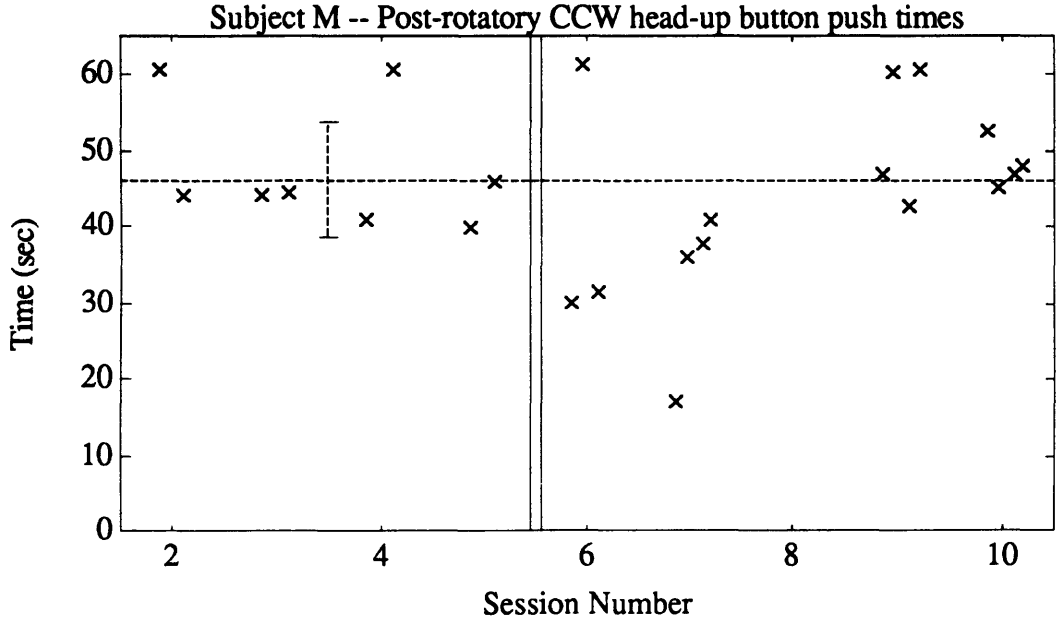


Figure 5.2b. Counter-clockwise head-up post-rotatory button push times for Subject M. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

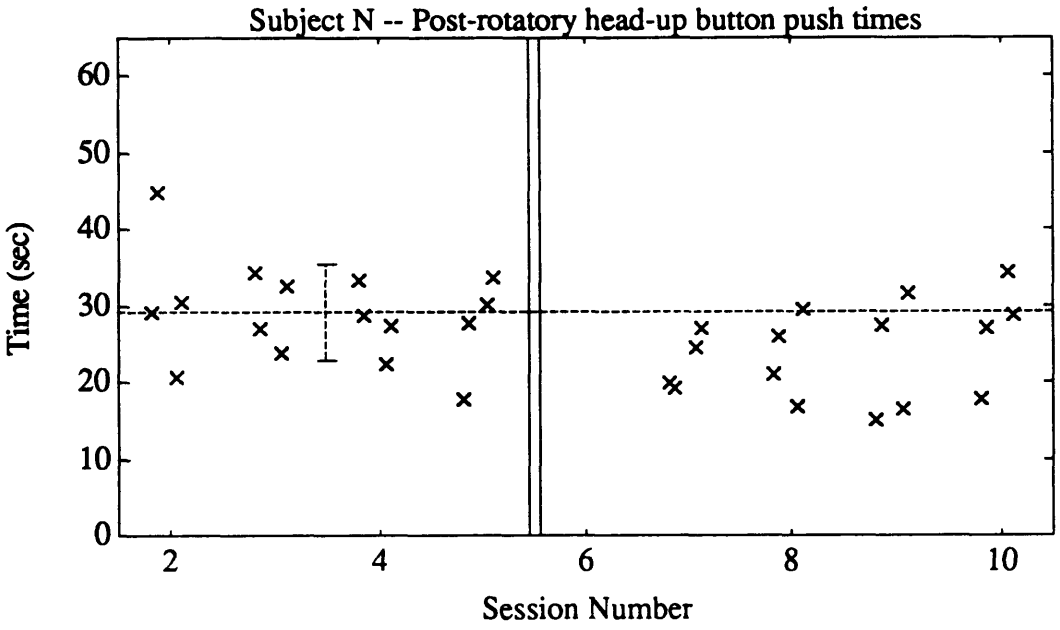


Figure 5.2c. Head-up post-rotatory button push times for Subject N. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

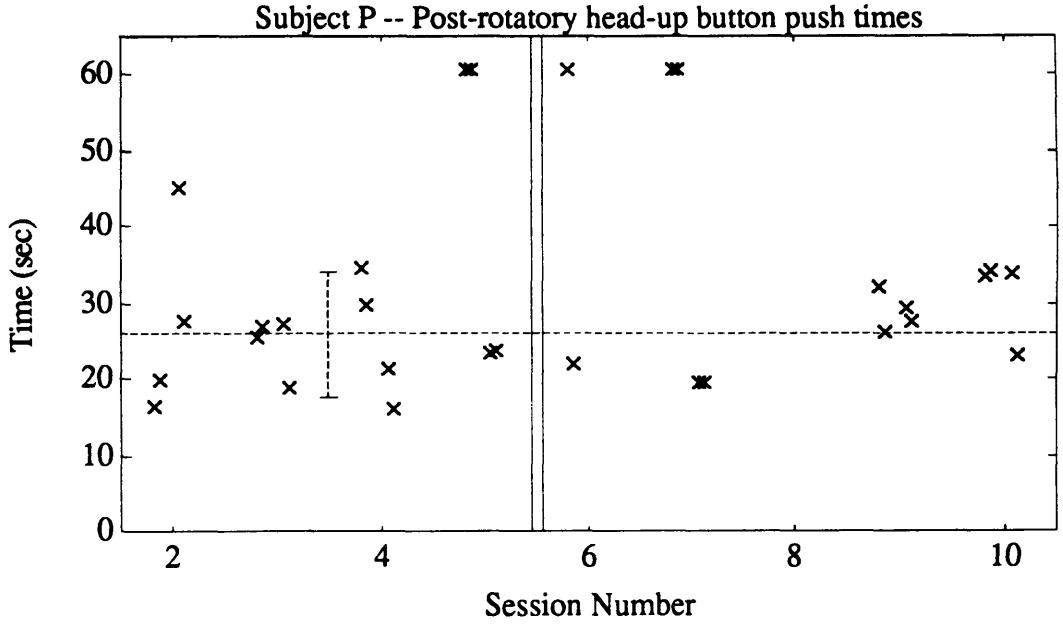


Figure 5.2d. Head-up post-rotatory button push times for Subject P. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

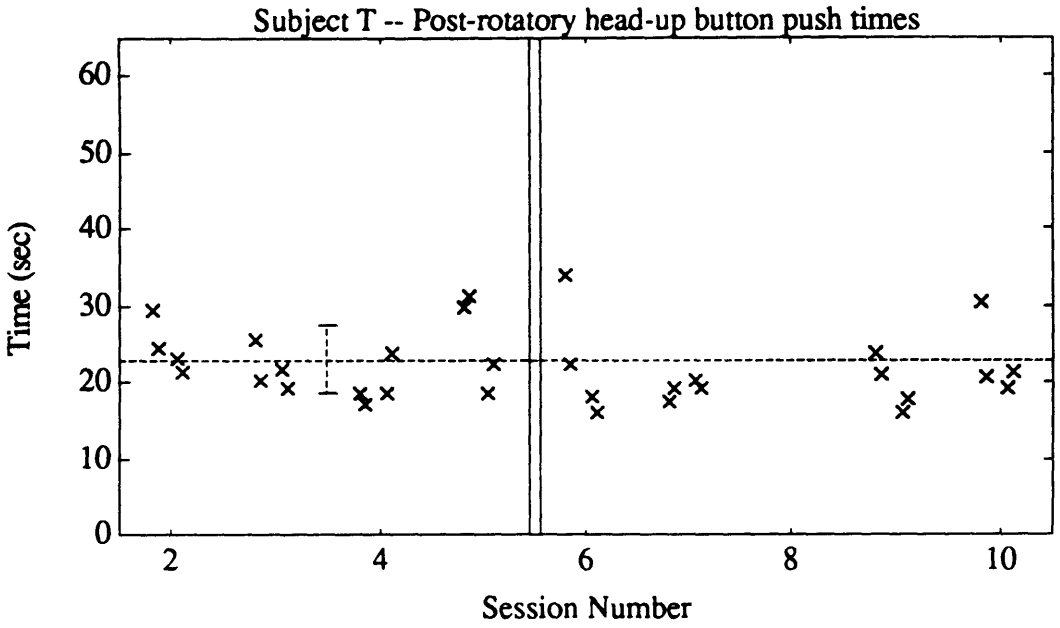


Figure 5.2e. Head-up post-rotatory button push times for Subject T. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

For subject M, the mean times were somewhat lower during the return sessions for both CW and CCW runs but not significantly lower as calculated by a t-test, possibly due in part to the large variance. The times were below average on 6 out of 8 CW runs and 6 out of 7 CCW runs, which was a significant difference ($p = 0.035$) according to a sign test. The times for both directions were somewhat greater, but not significantly, during the recovery sessions, being above average on 6 out of 8 CW runs and 6 out of 8 CCW runs.

For subject N, the times were significantly shorter during the return sessions and 7 out of 8 times were below average. During the recovery sessions, 6 out of 8 times were below average and the mean was still somewhat, but not significantly, reduced.

For subject P, the return times were as erratic as during BDC5. By the end of the post-flight week the responses had stabilized and the times were slightly, but not significantly, longer than the pre-flight mean. This increase was reflected in 7 out of 8 runs.

For subject T, the times were below average in 7 out of 8 runs in the return portion, and 6 out of 8 runs in the recovery portion. However, the mean time was not significantly different from the pre-flight mean.

5.2.3.3 Dumping T2 Changes

The return and recovery data was compared to the pre-flight data by F-tests for equal variances, and t-tests for differences in means. The test results are shown in Tables 5.13a and 5.13b for the return and recovery data respectively. If the variances of the two data sets were significantly different, the variance for the t-test was taken to be the larger of the two variances; otherwise, a pooled variance estimate was used. Figures 5.3a through 5.3c show graphically the raw data values, together with the mean (horizontal dashed line) and standard deviation (vertical dashed bars) of the pre-flight data. As was noted earlier, there were no dumping runs performed on subject M during the post-flight sessions.

Subject	F	df	p	var	df	t	p
N	3.95	(11,3)	> .2	48.19	14	1.856	< .1
P	1.45	(9,5)	> .2	15.56	14	.187	> .5
T	1.73	(10,7)	> .2	2.63	17	3.289	< .01

Table 5.13a. Summary of F-test and t-test results performed on post-flight (return) data for changes in the post-rotatory button push times for dumping runs. The columns shown are the same as in Table 5.6. A negative sign in the t-value indicates that the mean return dumping T2 time was larger than the mean pre-flight time.

Subject	F	df	p	var	df	t	p
N	5.04	(11,6)	< .1	41.14	17	1.360	> .1
P	1.38	(9,7)	> .2	15.38	16	.720	> .4
T	2.04	(10,7)	> .2	2.52	17	4.163	< .001

Table 5.13b. Summary of F-test and t-test results performed on post-flight (recovery) data for changes in the post-rotatory button push times for dumping runs. The columns shown are the same as in Table 5.6. A negative sign in the t-value indicates that the mean recovery dumping T2 time was larger than the mean pre-flight time.

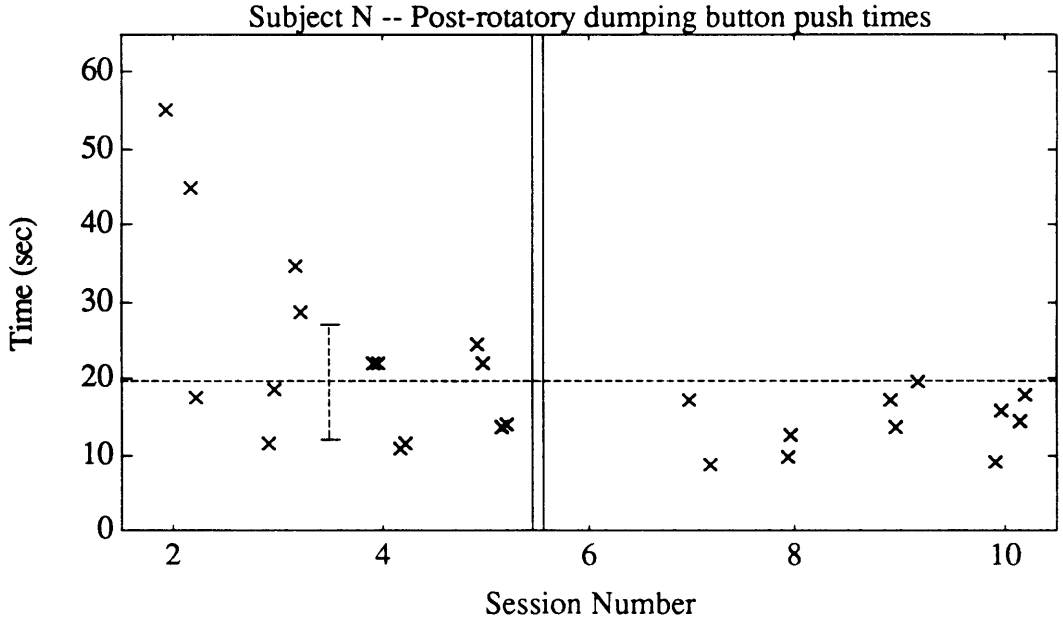


Figure 5.3a. Dumping post-rotatory button push times for Subject N. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

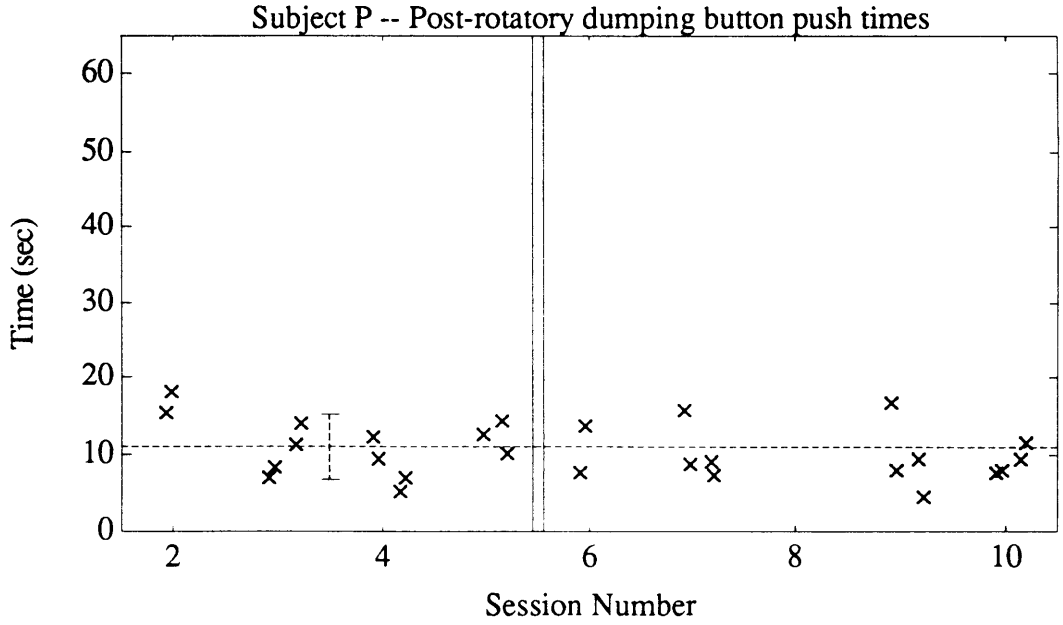


Figure 5.3b. Dumping post-rotatory button push times for Subject P. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

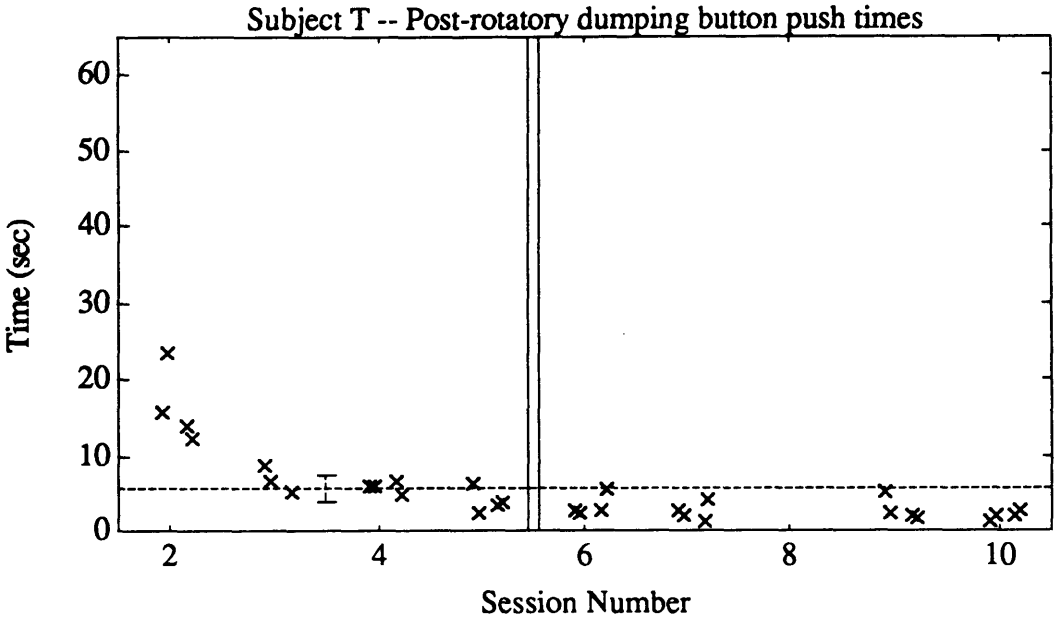


Figure 5.3c. Dumping post-rotatory button push times for Subject T. Individual data points are plotted with pre-flight mean (horizontal dashed line) and ±1 standard deviation (error bars). Vertical double bar divides pre-flight from post-flight data.

For subject N, a t-test indicated a trend ($p < .1$) to shorter times during the return phase followed by an increase to a slightly, but not significantly, reduced level during the recovery phase. This is supported by Figure 5.3a which shows that 10 out of 11 post-flight times were below average, and the remaining time was on the average ($p < 0.012$ by sign rank test).

For subject P, there was no appreciable change in the dumping T2 times. Even though the majority of the times (10 out of 14) were below average, the differences in the means were not at all significant; the magnitude of the differences was approximately one second and was therefore negligible.

For subject T, a t-test showed a significant reduction in the dumping T2 time after spaceflight. The means were significantly shorter during both the return and recovery portions, and 15 out of 16 post-flight times were below average ($p < .001$). However, there was also a decreasing trend in the pre-flight dumping T2 times, which made it hard to conclusively claim a real decrease. Additional subjective comments indicated that all sensation of rotation had disappeared by the time that the head movement was completed, unlike the pre-flight sessions; this supported the existence of a real difference.

5.2.3.4 Summary

The changes which occurred in the button push times from pre-flight to post-flight are summarized in Table 5.14. A statistically significant decrease or increase is indicated by a '--' or '++' respectively. A '-' or '+' sign is used to indicate a change that is convincing, even if not statistically significant. An empty cell means that there was no apparent change. In general, the button push times during the return sessions were either shorter or unchanged from the pre-flight sessions; by the end of the week, the times had increased partially or completely to the pre-flight norms.

Subject	Time	Return	Recovery
M	CW T1	- -	
	CCW T1		++
	CW h-up T2	-	
	CCW h-up T2	-	
N	T1	- -	
	h-up T2	- -	-
	dump T2	-	-
P	T1		
	h-up T2		
	dump T2		
T	T1		
	h-up T2		
	dump T2	- -	- -

Table 5.14. Summary of changes in button push times from pre-flight sessions to return and recovery sessions. The second column shows the range over which the times have been averaged. For example, "CW dump T2" indicates that only the T2 times from the CW dumping runs were combined, whereas "T1" indicates that T1 times were averaged across both directions and both head movement types.

5.3 Slow Phase Velocity

5.3.1 Calibration Factors

The corneo-retinal potential is known to vary over time, and is very sensitive to light level (Gonshor and Malcolm, 1971). This fluctuation is reflected in the change in the calibration factor to convert from measured A/D units to degrees of eye movement. There were nominally three calibrations performed on each subject during each session, and these conversion factors were linearly weighted to interpolate the calibration factors for each of the stimulus runs. The details of this procedure are described in Section 4.3.1. The horizontal calibration factors are tabulated in Tables 5.15a through 5.15d, and vertical calibration factors in Tables 5.16a through 5.16d. Calibration factors are expressed as a ratio of degrees of eye movement to number of measured A/D units. Missing data symbols (dashes) in the table indicate that either the run was not performed or the data was lost.

A surprising result was the presence of a consistent asymmetry in the horizontal calibration for subject T, wherein the rightward deflection (+10°) resulted in a voltage difference which was approximately 30% higher than the corresponding voltage difference from a leftward deflection (-10°). The medical history revealed that the subject had strabismus surgery when young, and suffered from esophoria and reduced acuity in the right eye. When the goggles were worn with the red filters for calibration, the left calibration target was usually not visible to the right eye. As such, the right eye may not have been properly positioned to reflect a leftward 10° deflection. The right target however was visible to the weak right eye and so it was properly positioned for rightward gaze deflection; even though the target was not visible to the normal left eye, it was also properly positioned. Since the EOG potential is the vector sum of the

corneo-retinal potential in each eye, perhaps this type of disconjugate eye movement could explain the asymmetry in the calibration. The calibration factor for this subject (as well as the others) was calculated using an average of both deflections. Since the rotation was in the dark and no targets were present, it was hoped that the weak eye would respond consistently.

The work by Gonshor and Malcolm analyzed the EOG potential in terms of μV/deg of eye movement, observing that it decreased for about ten minutes after normal white light was changed to red light for dark adaptation, increased for ten minutes further, and then held constant. There was no significant difference in calibration factors between red light and total darkness. Therefore, it was expected that the calibration factors in deg/unit (the inverse of Gonshor and Malcolm's) would first increase and then decrease over time. Out of the 30 cases where three calibration runs were performed, 14 followed this behaviour. There were 9 cases where the calibration factor reduced from the first calibration to the second, and from the second to the third; this was likely due to a longer time spent in dark adaptation before the calibration runs were performed. Of the remaining 7 cases (of which 4 were subject T), two exhibited increases from the first to second, and second to third calibrations, while the other 5 showed decreases followed by a slight increase; these differences were likely due to either the noise in the calibrations, or to random variation in the behaviour of the adaptation process.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
1	**.0549**	**.0674**	**.0684**	**.0611**	**.0536**	**.0797**	**.0591**	**.0607**
2	.0565	.0688	.0686	.0617	.0542	.0786	.0588	.0606
3	.0595	.0717	.0690	.0630	.0553	.0765	.0583	.0603
4	.0626	.0745	.0693	.0642	.0565	.0743	.0578	.0600
5	.0657	.0774	.0697	.0655	.0576	.0722	.0573	.0597
6	**.0673**	**.0788**	**.0699**	**.0661**	**.0582**	**.0711**	**.0571**	**.0596**
7	.0659	.0775	.0707	.0651	.0571	.0718	.0568	.0595
8	.0630	.0750	.0722	.0630	.0549	.0731	.0563	.0592
9	.0601	.0726	.0738	.0609	.0526	.0745	.0559	.0590
10	.0572	.0701	.0753	.0588	—	.0758	.0554	.0587
11	**.0558**	**.0688**	**.0761**	**.0578**	**.0515**	**.0765**	**.0551**	**.0586**

Table 5.15a. Horizontal calibration factors for Subject M, in degrees of eye movement per measured A/D unit. Runs 1, 6, and 11 were calibration runs; factors for other runs were interpolated from these values.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
1	**.0226**	**.0277**	**.0187**	**.0203**	**.0304**	**.0270**	**.0251**	**.0209**
2	.0229	.0273	.0189	.0205	.0302	.0263	.0248	.0218
3	.0235	.0265	.0193	.0210	.0298	.0250	.0241	.0237
4	.0241	.0258	.0196	.0214	.0294	.0236	.0234	.0255
5	.0247	.0250	.0200	.0219	.0290	.0223	.0227	.0274
6	**.0250**	**.0246**	**.0202**	**.0221**	**.0288**	**.0216**	**.0224**	**.0283**
7	.0248	.0240	.0197	.0217	.0280	.0212	.0221	.0277
8	.0244	.0228	.0188	.0210	.0265	.0205	.0215	.0263
9	.0240	.0217	.0178	.0202	.0250	—	.0209	.0250
10	.0236	.0205	.0169	.0195	—	—	—	.0237
11	**.0234**	**.0199**	**.0164**	**.0191**	**.0242**	**.0202**	**.0206**	**.0231**

Table 5.15b. Horizontal calibration factors for Subject N, in degrees of eye movement per measured A/D unit. Runs 1, 6, and 11 were calibration runs; factors for other runs were interpolated from these values.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
1	**.0369**	**.0427**	**.0485**	**.0404**	**.0341**	**.0334**	**.0456**	—
2	.0358	.0426	.0490	.0405	.0344	.0328	.0459	.0357
3	.0337	.0424	.0500	.0406	.0351	.0314	.0464	.0357
4	.0315	.0422	.0511	.0407	.0357	.0301	.0469	.0357
5	.0294	.0420	.0521	.0408	.0364	.0289	.0474	.0357
6	**.0283**	**.0419**	**.0526**	**.0409**	**.0367**	**.0282**	**.0476**	**.0357**
7	.0278	.0407	.0509	.0400	—	.0284	.0459	.0347
8	.0269	.0384	.0476	.0382	—	.0288	.0423	.0326
9	—	.0361	.0443	.0365	—	.0292	.0388	.0305
10	—	.0338	.0410	.0347	—	.0296	.0353	.0284
11	**.0264**	**.0326**	**.0393**	**.0338**	—	**.0298**	**.0336**	**.0274**

Table 5.15c. Horizontal calibration factors for Subject P, in degrees of eye movement per measured A/D unit. Runs 1, 6, and 11 were calibration runs; factors for other runs were interpolated from these values.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
1	**.0633**	**.0526**	**.0512**	**.0533**	**.0591**	**.0447**	**.0352**	**.0411**
2	.0610	.0511	.0520	.0520	.0571	.0464	.0362	.0416
3	.0563	.0481	.0536	.0495	.0531	.0499	.0382	.0426
4	.0516	.0450	.0552	.0470	.0492	.0534	.0403	.0435
5	.0469	.0420	.0568	.0445	.0452	.0569	.0423	.0445
6	**.0446**	**.0405**	**.0576**	**.0433**	**.0432**	**.0586**	**.0433**	**.0450**
7	.0451	.0401	.0566	.0436	.0434	.0567	.0428	.0457
8	.0462	.0394	.0545	.0441	.0438	.0529	.0418	.0471
9	.0473	.0387	.0524	.0446	.0441	.0490	.0407	.0486
10	.0484	—	.0503	.0451	.0445	.0452	.0397	.0500
11	**.0489**	**.0384**	**.0493**	**.0454**	**.0447**	**.0433**	**.0392**	**.0507**

Table 5.15d. Horizontal calibration factors for Subject T, in degrees of eye movement per measured A/D unit. Runs 1, 6, and 11 were calibration runs; factors for other runs were interpolated from these values.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
1	**.0392**	**.0648**	**.0576**	**.0519**	**.0450**	**.0637**	**.0521**	**.0620**
2	.0402	.0657	.0579	.0525	.0464	.0627	.0518	.0620
3	.0422	.0676	.0586	.0536	.0492	.0608	.0513	.0620
4	.0443	.0695	.0592	.0547	.0521	.0589	.0507	.0620
5	.0463	.0714	.0599	.0558	.0549	.0570	.0502	.0620
6	**.0473**	**.0723**	**.0602**	**.0564**	**.0563**	**.0560**	**.0499**	**.0620**
7	.0477	.0709	.0590	.0557	.0557	.0561	.0498	.0621
8	.0486	.0681	.0565	.0543	.0544	.0563	.0495	.0625
9	.0494	.0652	.0541	.0530	.0531	.0564	.0493	.0628
10	.0503	.0624	.0516	.0516	—	.0566	.0490	.0630
11	**.0507**	**.0610**	**.0504**	**.0509**	**.0525**	**.0567**	**.0489**	**.0632**

Table 5.16a. Vertical calibration factors for Subject M, in degrees of eye movement per measured A/D unit. Runs 1, 6, and 11 were calibration runs; factors for other runs were interpolated from these values.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
1	**.0596**	**.0535**	**.0450**	**.0521**	**.0495**	**.0405**	**.0476**	**.0393**
2	.0594	.0522	.0457	.0528	.0491	.0395	.0472	.0405
3	.0589	.0495	.0472	.0542	.0484	.0374	.0463	.0428
4	.0585	.0469	.0486	.0555	.0476	.0353	.0454	.0451
5	.0580	.0442	.0501	.0569	.0469	.0332	.0445	.0474
6	**.0578**	**.0429**	**.0508**	**.0576**	**.0465**	**.0322**	**.0441**	**.0486**
7	.0584	.0431	.0502	.0564	.0447	.0314	.0433	.0493
8	.0596	.0434	.0490	.0540	.0411	.0297	.0415	.0507
9	.0607	.0437	.0477	.0516	.0375	—	.0399	.0521
10	.0619	.0440	.0465	.0492	—	—	—	.0535
11	**.0625**	**.0442**	**.0459**	**.0480**	**.0357**	**.0289**	**.0390**	**.0542**

Table 5.16b. Vertical calibration factors for Subject N, in degrees of eye movement per measured A/D unit. Runs 1, 6, and 11 were calibration runs; factors for other runs were interpolated from these values.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
1	**.0523**	**.0422**	**.0555**	**.0542**	**.0619**	**.0576**	**.0505**	—
2	.0525	.0425	.0558	.0540	.0626	.0572	.0503	.0584
3	.0529	.0430	.0563	.0535	.0640	.0564	.0499	.0584
4	.0533	.0435	.0568	.0530	.0654	.0557	.0495	.0584
5	.0537	.0440	.0573	.0525	.0668	.0549	.0491	.0584
6	**.0539**	**.0443**	**.0576**	**.0523**	**.0675**	**.0545**	**.0489**	**.0584**
7	.0524	.0433	.0578	.0519	—	.0546	.0472	.0565
8	.0495	.0413	.0580	.0513	—	.0547	.0439	.0528
9	—	.0393	.0583	.0505	—	.0548	.0405	.0490
10	—	.0373	.0586	.0499	—	.0549	.0372	.0453
11	**.0480**	**.0363**	**.0588**	**.0495**	—	**.0550**	**.0355**	**.0434**

Table 5.16c. Vertical calibration factors for Subject P, in degrees of eye movement per measured A/D unit. Runs 1, 6, and 11 were calibration runs; factors for other runs were interpolated from these values.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
1	**.1053**	**.0957**	**.0810**	**.0790**	**.0636**	**.0633**	**.0583**	**.0733**
2	.0999	.0956	.0815	.0798	.0624	.0684	.0579	.0761
3	.0891	.0954	.0826	.0813	.0600	.0786	.0570	.0817
4	.0783	.0953	.0837	.0829	.0575	.0888	.0562	.0872
5	.0675	.0951	.0848	.0844	.0551	.0990	.0553	.0928
6	**.0621**	**.0950**	**.0853**	**.0852**	**.0539**	**.1041**	**.0549**	**.0956**
7	.0625	.0922	.0842	.0855	.0544	.1001	.0561	.0974
8	.0633	.0866	.0820	.0860	.0554	.0922	.0585	.1010
9	.0642	.0809	.0797	.0866	.0565	.0842	.0608	.1046
10	.0650	—	.0775	.0871	.0575	.0763	.0632	.1082
11	**.0654**	**.0781**	**.0764**	**.0874**	**.0580**	**.0723**	**.0644**	**.1100**

Table 5.16d. Vertical calibration factors for Subject T, in degrees of eye movement per measured A/D unit. Runs 1, 6, and 11 were calibration runs; factors for other runs were interpolated from these values.

5.3.2 Discarded Runs

A number of runs were discarded due to the conditions outlined in Section 4.7. In some cases, the entire run was discarded, while in others only the per-rotatory or post-rotatory portion was rejected. Tables 5.17a through 5.17d show the rejection status of the runs, namely which portions of which runs were discarded. A "1" (resp. "2") means that the per-rotatory (resp. post-rotatory) section was eliminated; therefore, a "12" means that the entire run was eliminated. An empty cell means that the entire run was kept. A run which was not performed, as indicated in Section 5.1, is shown by a "—".

In addition to these rejections, five runs required some minor editing. M203 and P202 both contained large sections of data where the EOG had drifted outside the range of the A/D converter; these sections had been missed by the outlier detection algorithm but were manually marked as bad data. T302 and T405 suffered computer crashes in the middle of the run, but the data acquisition was recommenced in time to get information for both the per- and post-rotatory portions. The intervening data were manually filled with dummy values and marked as bad data, so that the good data could be used. Finally, N208 contained a significant reduction in the SPV for the first 2.5 seconds of the per-rotatory peak. While the SPV was reduced, it was not low enough to be classified as a dropout by the automated algorithm. However, the repeatability of the data for N was such that this was clearly a dropout. This was an isolated case. No similar editing was performed on any other subjects, or on any other runs for N.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2			1	2	1	1 2	1 2	
3			1 2	1 2	1 2	1 2		
4			1 2	1		1 2	1	
5		1 2	1 2	1 2	2	1 2		
7		1 2	1 2		1 2	1 2		
8			1	1	1 2	1 2	1	2
9				2	1	1 2	1	
10		2	1 2		—	1 2		

Table 5.17a. Rejection status of runs for Subject M. Explanation of symbols in text.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2								
3								
4								
5								
7								
8								
9						—		
10					—	—	—	

Table 5.17b. Rejection status of runs for Subject N. Explanation of symbols in text.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2			1			1 2		1 2
3			1 2		2	1 2		
4			1 2	1 2		1 2	1 2	
5		2	1 2	2	2	2	2	2
7		1	1 2	2	—	1 2		
8		2	2	1 2	—	1 2	1	
9	—		1 2	2	—		1 2	1
10	—	2	1 2		—			

Table 5.17c. Rejection status of runs for Subject P. Explanation of symbols in text.

Run	BDC2	BDC3	BDC4	BDC5	BDC6	BDC7	BDC9	BDC10
2			1			2	1 2	
3					2			
4	2		1			1	1	
5		2		1 2				
7			1				2	2
8		2						
9		2		1			1	
1 0		—						

Table 5.17d. Rejection status of runs for Subject T. Explanation of symbols in text.

5.3.3 Pre-flight Baseline SPV Response

The "good data" portions of the pre-flight runs were combined to calculate means and variances for the baseline response for each subject to each stimulus type. These average profiles were then analyzed for differences between CW and CCW responses, between head-up and dumping responses, and between per- and post-rotatory head-up responses. Since the SPV had in general decayed to nearly zero after about 40 seconds, only the first 40 seconds of the per- and post-rotatory SPV profiles were included in the comparisons. The first second after the start or stop of the chair was also excluded since the SPV changed rapidly during those portions.

Since the per-rotatory stimulus was identical for head-upright and dumping runs, no difference was expected between the corresponding SPV profiles. Indeed, the average responses appeared similar. Therefore, the per-rotatory portions of both head-upright and dumping runs were combined for all subsequent analysis.

The vertical SPV data was not analyzed in detail, since the nature of the experiment was to stimulate the horizontal semicircular canals. However, several runs (incorporating all subject and all stimuli types) were spot-checked to determine if there was any vertical nystagmus present. In general, there was no vertical nystagmus. A couple of runs exhibited a low level of nystagmus (< 5 °/s) which correlated with the horizontal nystagmus in relative magnitude and duration. This supposed "vertical" nystagmus was probably due to misalignment of the electrodes which produced a projection of the horizontal eye position on the vertical channel, or perhaps due to the subject not holding the head perfectly upright during the rotation. Most of the runs produced no vertical nystagmus.

Of particular note was the lack of vertical nystagmus after dumping head movements. Many investigators believe that the decreased level of horizontal nystagmus is due to a shift in the axis about which the nystagmus occurs, resulting in a buildup of either vertical or torsional nystagmus. Theoretically, the dumping head movement in the pitch plane would shift the nystagmus to the torsional axis. Torsional eye position can not be measured by EOG; so, nothing can be said about the presence of torsional nystagmus. However, there was no buildup of vertical nystagmus, which was consistent with theory.

5.3.3.1 Directional Asymmetry

The responses of all four subjects were checked for a directional asymmetry. The per-rotatory portions of the CW runs were compared to the per-rotatory portions of the CCW runs; the post-rotatory portions of the CW head-up runs were compared to the post-rotatory portions of the CCW head-up runs; and the post-rotatory portions of the

CW dumping runs were compared to the post-rotatory portions of the CCW dumping runs.

The results are shown in Table 5.18. The number of degrees of freedom ("dof") is the number of time points at which the data were compared; "r" is the ratio of the test statistic λ to the number of degrees of freedom; "p10" and "p05" are the percentage points for 10% and 5% significance in "r", as interpolated from Tables 4.2 and 4.3; "N1" and "N2" are the average sample sizes for the average CW and CCW respectively, from which "p10" and "p05" were estimated. Ratios which are statistically significant ($p < 0.05$) are shown in boldface type. Ratios which trend towards significance ($p < 0.10$) are underlined.

Subject	Time	N1	N2	dof	r	p10	p05
M	per	9.01	5.99	155	**1.89**	1.56	1.64
	h-up post	3.98	5.01	144	1.39	1.89	2.06
	dump post	4.85	2.63	115	1.88	2.31	2.58
N	per	15.47	15.42	156	**1.68**	1.40	1.46
	h-up post	6.78	7.29	156	0.34	1.54	1.63
	dump post	7.25	6.71	156	1.06	1.56	1.65
P	per	8.10	8.04	156	0.93	1.50	1.55
	h-up post	5.86	2.39	153	1.41	2.04	2.21
	dump post	2.69	1.81	107	1.87	7.76	8.01
T	per	9.13	10.46	156	**4.69**	1.45	1.52
	h-up post	6.29	5.44	154	**4.82**	1.68	1.76
	dump post	5.04	3.84	154	**2.29**	1.91	2.06

Table 5.18. Sum of t-squares tests for directional asymmetry within pre-flight data. The "Time" column refers to the portion of the SPV profiles which were compared: per- or post-rotatory, head-up or dumping. Explanation of other entries in text.

For subject M, there was a significant asymmetry in the per-rotatory response, with the CCW stimulus evoking a higher response than the CW stimulus (see Figure 5.4a). This asymmetry was opposite to the asymmetry in the button push times described earlier. There were no significant differences in the post-rotatory responses, even though the CW runs (relatively speaking, a CCW stimulus) appeared to produce a slightly higher response than the CCW runs; this may be due to the small N_1 and N_2. Overall, the asymmetry appears to be real.

For subject N, the difference between the per-rotatory responses to CW and CCW stimuli was significant. However, the responses were only significantly different near the end of the profiles (after 30 seconds), where the CCW SPV was less than the CW response; a comparison between the first 30 seconds of data showed no significant difference. This was indicative of a lower velocity storage component in the CCW response. There were no significant differences between the post-rotatory responses, either head-up or dumping, and in fact the ratios for these tests were below the expected values. Therefore, it was concluded that there was no real directional asymmetry.

Subject P exhibited no significant directional asymmetry under any of the conditions, and none was apparent upon visual inspection of the profiles.

Subject T demonstrated a very large directional asymmetry (see Figures 5.4b through 5.4d), with the per-rotatory and head-up post-rotatory differences being significant at the $p < 0.001$ level. The SPV responses to CCW stimuli (CCW per-rotatory or CW post-rotatory) were similar to those of the other three subjects; however, the SPV response to a CW stimulus was much lower in magnitude.

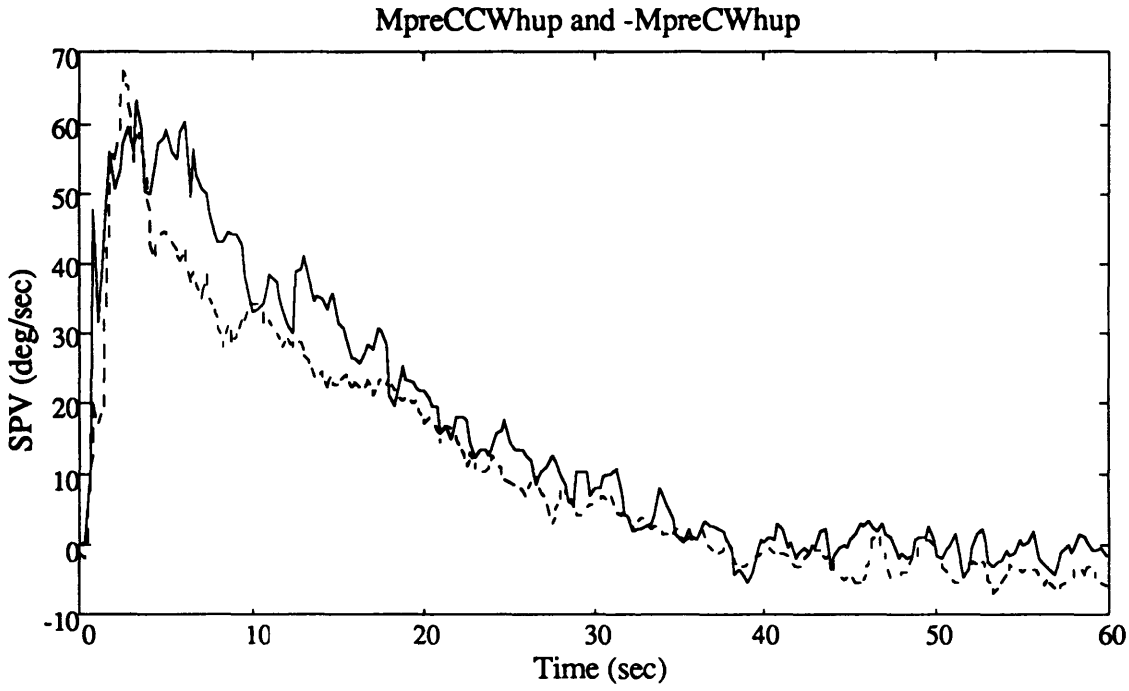


Figure 5.4a. Directional asymmetry in subject M, pre-flight, per-rotatory. CCW response (solid line) and negative of CW response (dashed line) are shown.

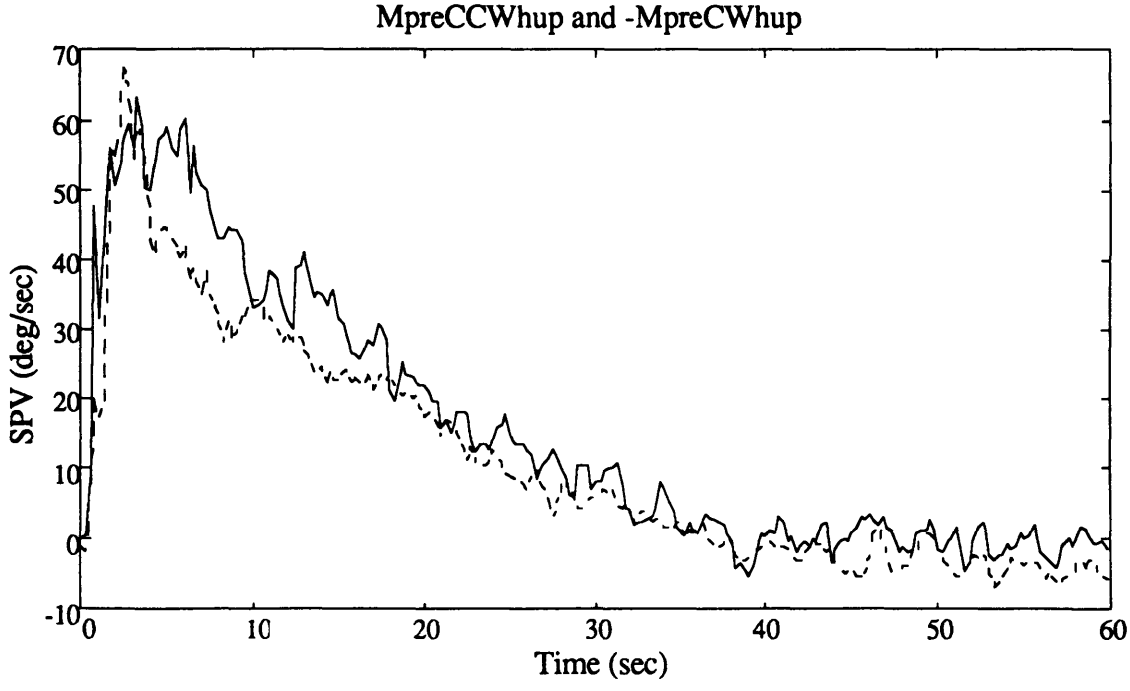


Figure 5.4b. Directional asymmetry in subject T, pre-flight, per-rotatory. CCW response (solid line) and negative of CW response (dashed line) are shown.

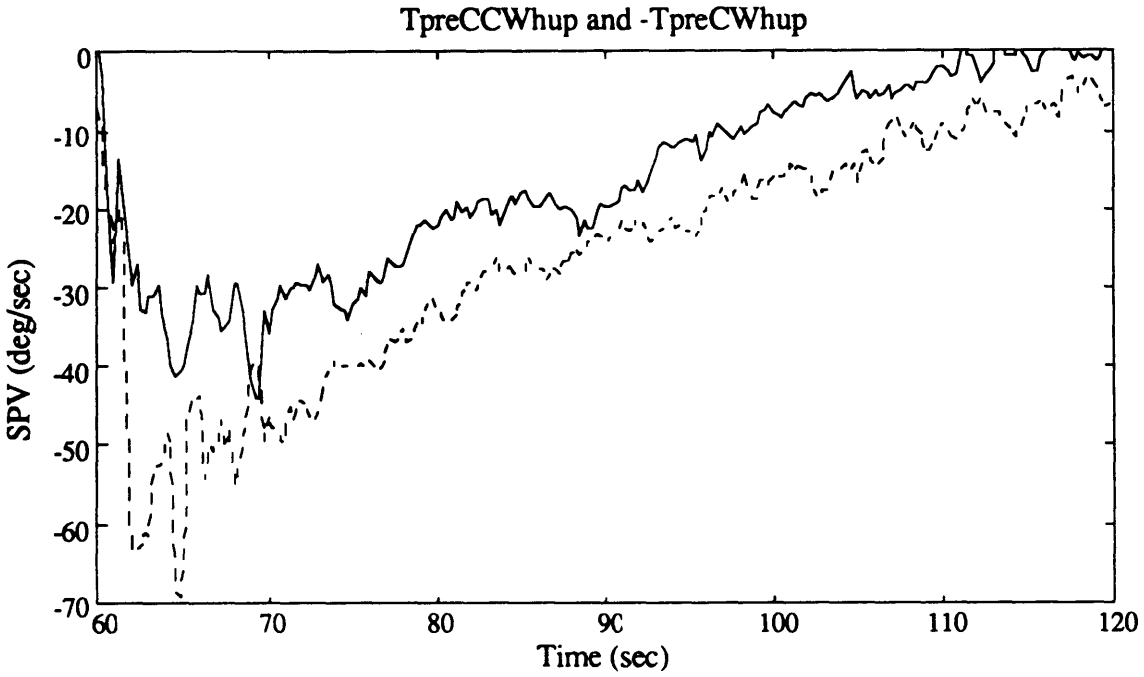


Figure 5.4c. Directional asymmetry in subject T, pre-flight, head-up post-rotatory. CCW response (solid line) and negative of CW response (dashed line) are shown.

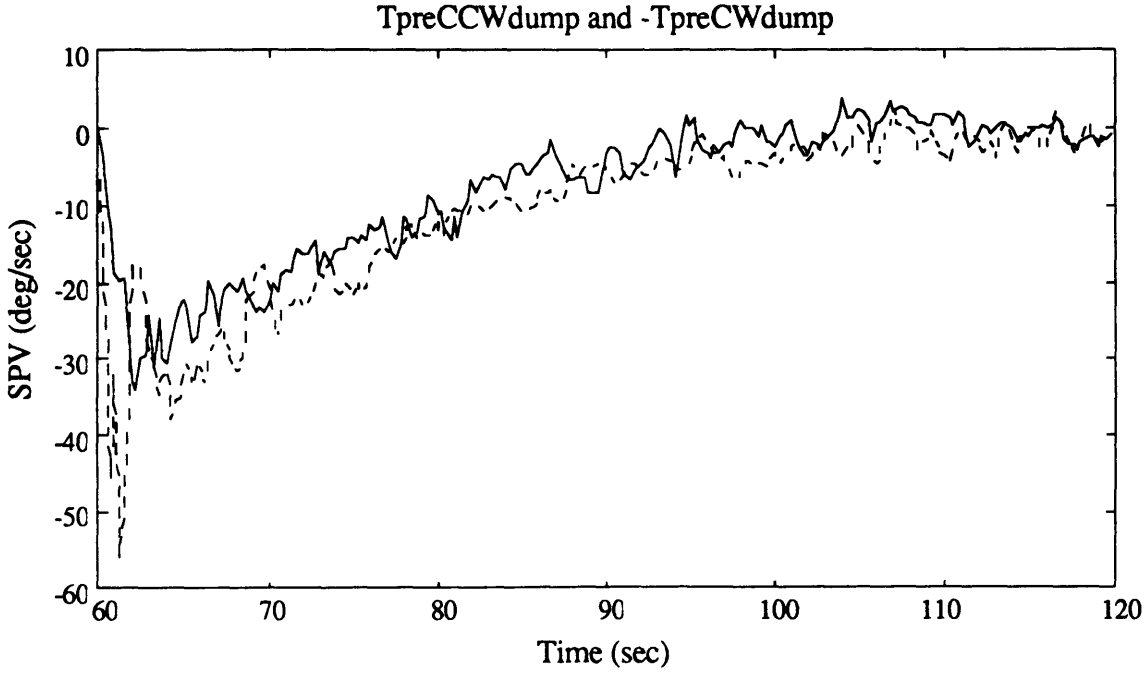


Figure 5.4d. Directional asymmetry in subject T, pre-flight, dumping post-rotatory. CCW response (solid line) and negative of CW response (dashed line) are shown.

5.3.3.2 Per-rotatory vs. Post-rotatory

Comparisons were also made to determine if the per-rotatory and post-rotatory SPV responses were different, as would be predicted by the neural adaptation term in the model. The CW per-rotatory portions of the SPV profiles were compared to the CCW post-rotatory portions, and the CCW per-rotatory portions were compared to the CW post-rotatory portions; this preserves the direction of the relative stimulus. The results are summarized in Table 5.19, with the columns as described in Section 5.3.3.1.

In three of the subjects (N, P, and T), the post-rotatory SPV was significantly greater in magnitude than the per-rotatory SPV (see Figures 5.5a through 5.5c). The fourth subject did not show any significant difference. These differences between the per- and post-rotatory SPV were, in general, fairly constant in magnitude and were particularly significant as the magnitude of the SPV decreased.

All four subjects demonstrated reversals in the direction of the SPV during the per-rotatory portions, presumably caused by neural adaptation. This reversal would introduce an offset term to the post-rotatory response, producing a difference similar to that seen. It is therefore likely that these differences were due largely to neural adaptation.

Subject	Direction	N1	N2	dof	r	p10	p05
M	CW	9.26	4.93	148	1.26	1.55	1.63
	CCW	6.18	3.96	145	1.76	1.77	1.90
N	CW	15.47	7.29	156	**2.10**	1.39	1.44
	CCW	15.42	6.78	156	**6.54**	1.40	1.45
P	CW	8.14	2.39	153	1.40	1.73	1.83
	CCW	8.04	5.83	156	**2.40**	1.59	1.68
T	CW	9.21	5.44	154	**4.47**	1.55	1.64
	CCW	10.46	6.23	156	**2.84**	1.51	1.59

Table 5.19. Sum of t-squares tests for differences between per- and post-rotatory responses within pre-flight data. The "Direction" column refers to the direction of the stimulus to the semi-circular canals. Explanation of other entries in text.

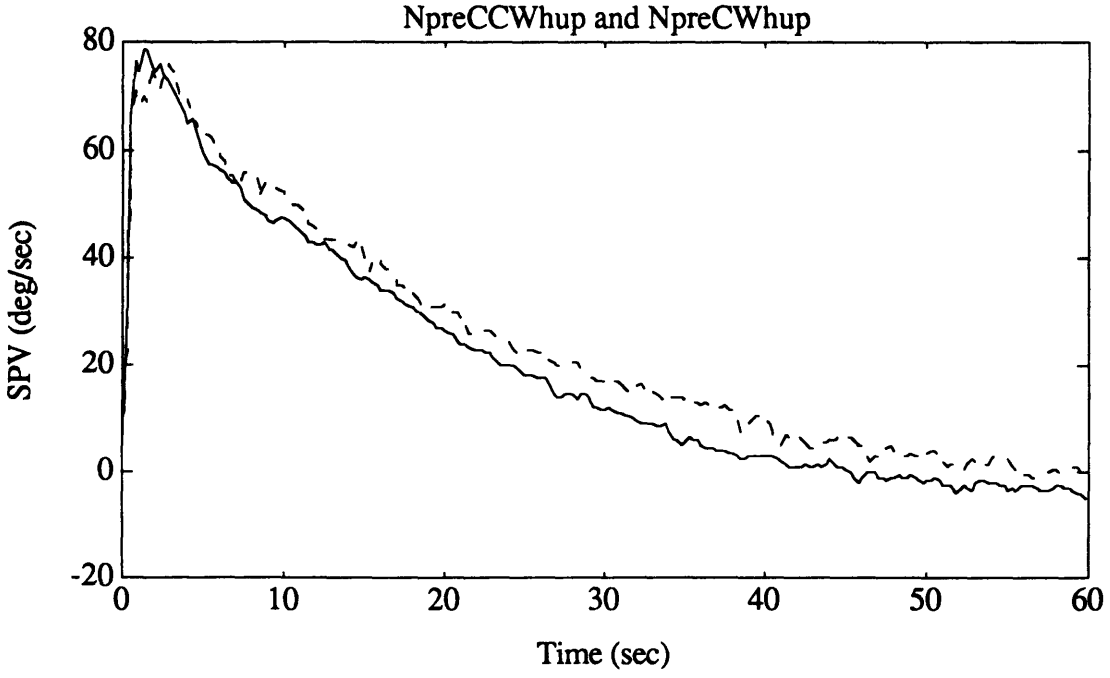


Figure 5.5a. Per/post-rotatory asymmetry in subject N, pre-flight, CCW. CCW per-rotatory (solid line) and CW post-rotatory (dashed line) are shown.

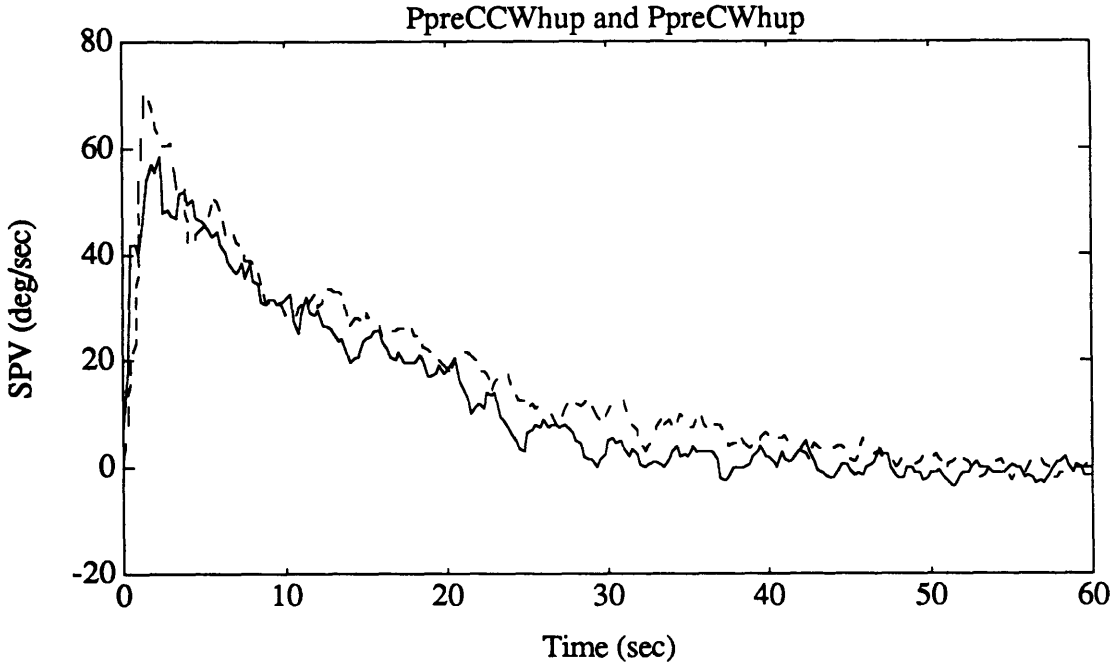


Figure 5.5b. Per/post-rotatory asymmetry in subject P, pre-flight, CCW. CCW per-rotatory (solid line) and CW post-rotatory (dashed line) are shown.

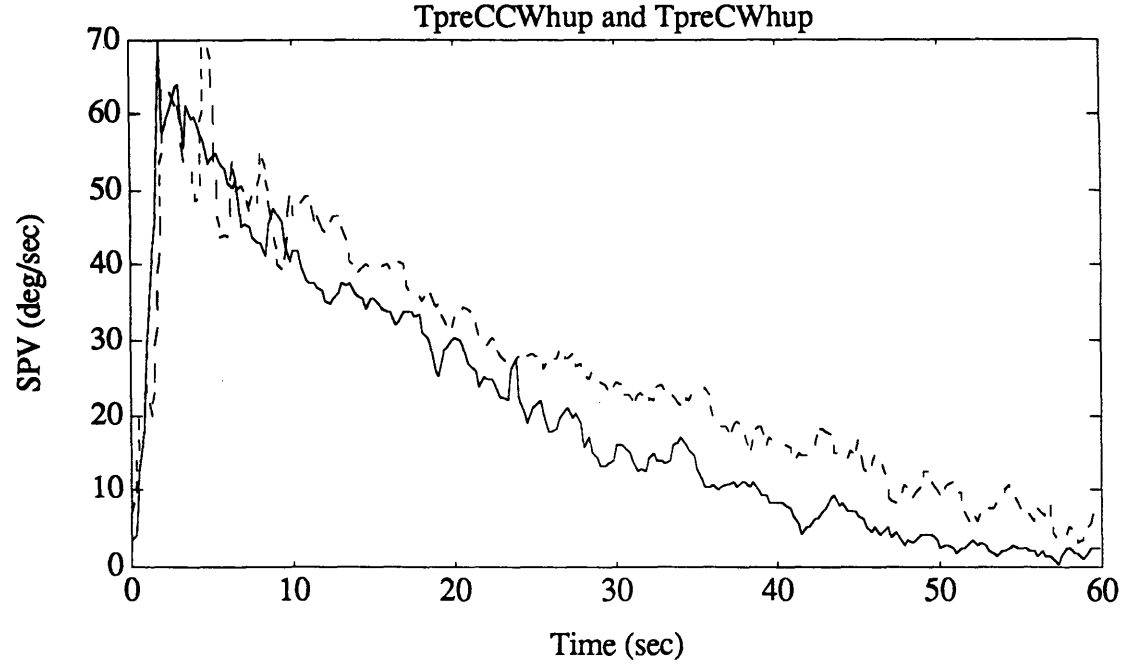


Figure 5.5c. Per/post-rotatory asymmetry in subject T, pre-flight, CCW. CCW per-rotatory (solid line) and CW post-rotatory (dashed line) are shown.

5.3.3.3 Dumping Effects

Finally, comparisons were made between the head-up and dumping post-rotatory responses to determine the effects of the dumping head movement. The results are summarized in Table 5.20, with the columns as described in Section 5.3.3.2. CW and CCW responses were tested separately. The direction shown in the table is that of the chair rotation (per-rotatory stimulus); it should be emphasized that the relative stimulus to the semicircular canals, evoking the post-rotatory responses, was in the opposite direction to the per-rotatory stimulus.

As expected, the dumping head movement produced significant reductions in the magnitude of the SPV in both directions for all subjects. Figures 5.6a through 5.6d show the differences in the SPV of subjects M and N.

Subject	Direction	N1	N2	dof	r	p10	p05
M	CW	4.03	4.54	127	**6.85**	1.93	2.09
	CCW	5.09	2.47	128	**6.02**	2.30	2.57
N	CW	6.78	7.25	156	**33.85**	1.54	1.63
	CCW	7.29	6.71	156	**39.02**	1.56	1.65
P	CW	6.04	2.46	136	**8.65**	1.97	2.11
	CCW	2.57	1.77	115	**10.31**	8.20	8.42
T	CW	6.29	5.04	154	**20.75**	1.70	1.78
	CCW	5.49	3.88	152	**10.12**	1.85	1.99

Table 5.20. Sum of t-squares tests for differences between head-upright and dumping post-rotatory responses within pre-flight data. The "Direction" column refers to the direction of the (per-rotatory) chair rotation. Explanation of other entries in text.

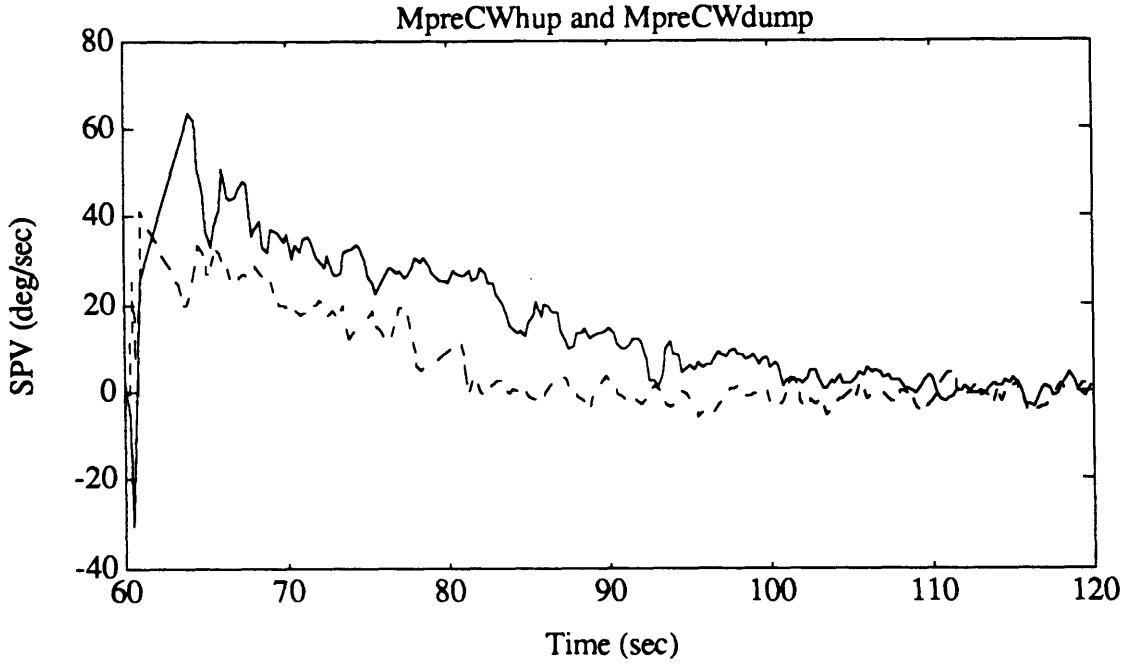


Figure 5.6a. Head-up and dumping post-rotatory SPV in subject M, pre-flight, CW. CW post-rotatory head-up (solid line) and dumping (dashed line) SPV are shown.

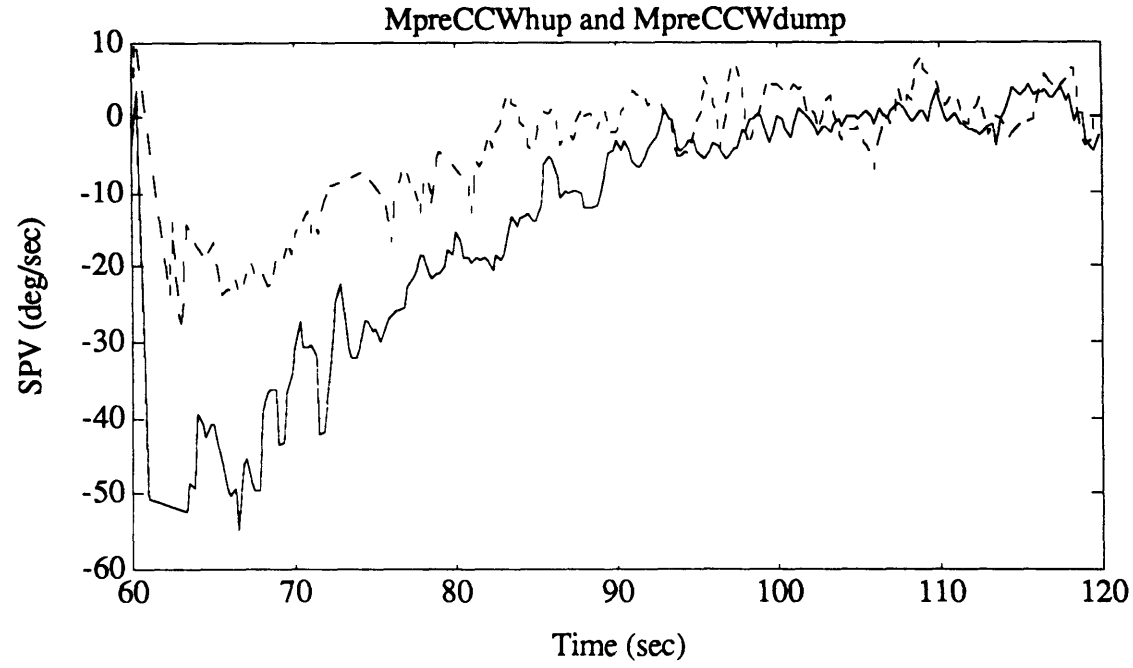


Figure 5.6b. Head-up and dumping post-rotatory SPV in subject M, pre-flight, CCW. CCW post-rotatory head-up (solid line) and dumping (dashed line) SPV are shown.

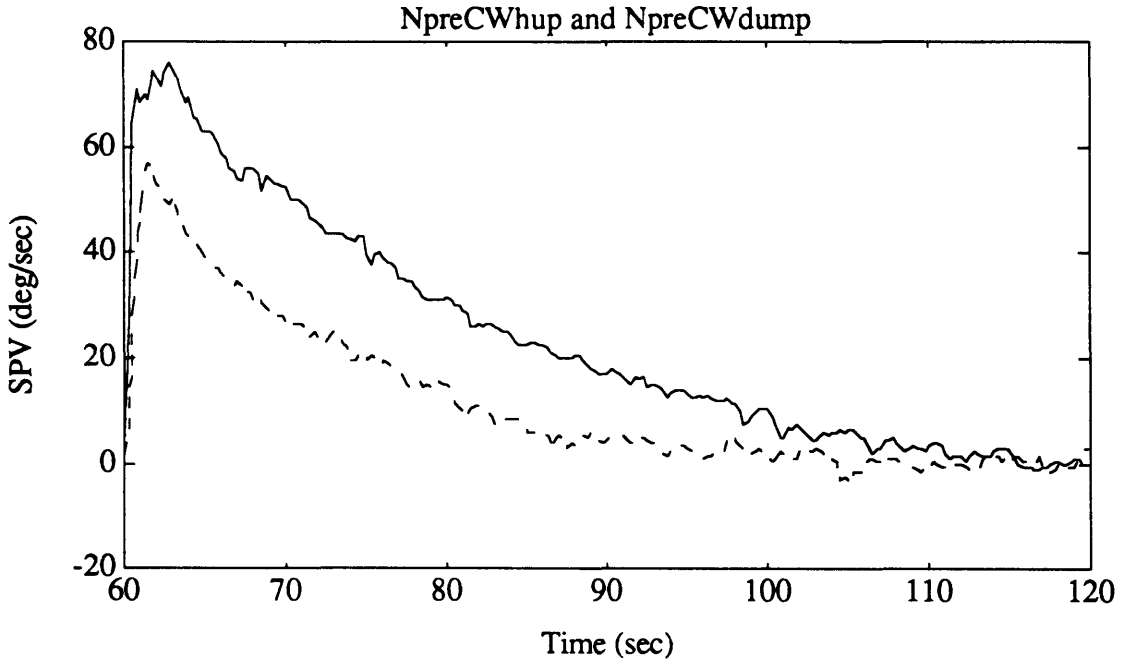


Figure 5.6c. Head-up and dumping post-rotatory SPV in subject N, pre-flight, CW. CW post-rotatory head-up (solid line) and dumping (dashed line) SPV are shown.

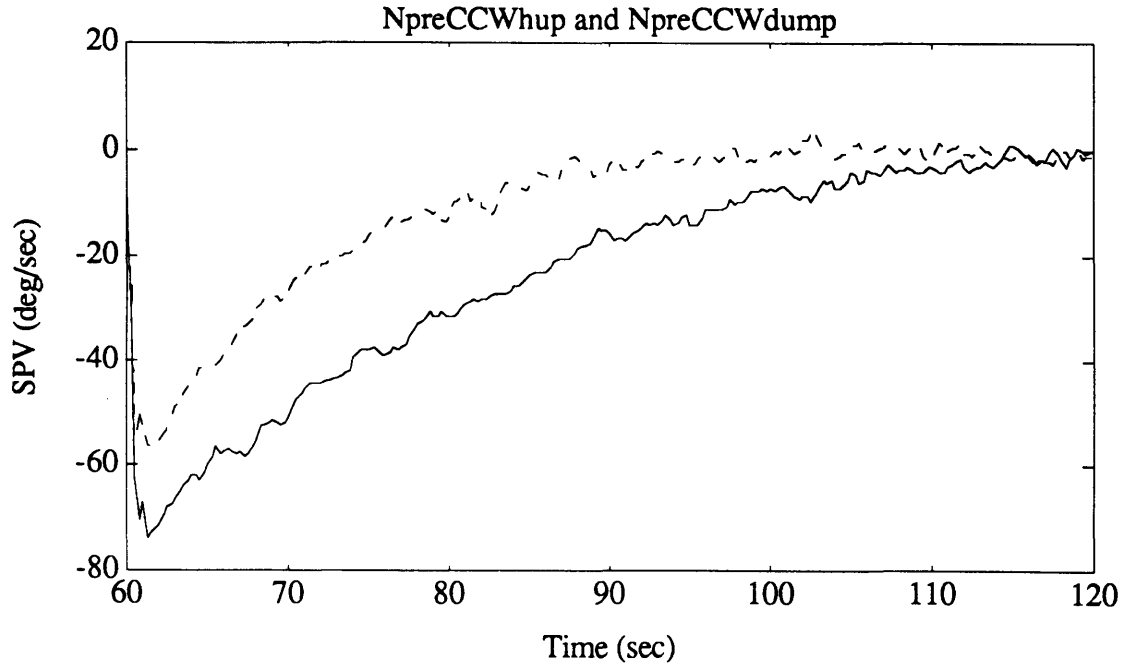


Figure 5.6d. Head-up and dumping post-rotatory SPV in subject N, pre-flight, CCW. CW post-rotatory head-up (solid line) and dumping (dashed line) SPV are shown.

5.3.4 Post-flight SPV

The post-flight data was averaged similarly to the pre-flight data. The same comparisons which were made within the pre-flight data were also made within the return data. Then, comparisons were made between pre-flight and return SPV responses, and between pre-flight and recovery responses. All comparisons were made with the sum of t-squares test, using the first 40 seconds (excluding the first second) of the per- and post-rotatory SPV.

The quality of data from subjects M and P was so poor, as indicated in Tables 5.17a and 5.17c, that no SPV data was analyzed for the return sessions. As mentioned earlier, there were also no dumping runs performed on subject M.

The pre-flight and recovery SPV profiles for subject M are shown in Figures 5.7a through 5.7d. Per- and post-rotatory responses are shown separately, with the scales standardized. No dumping runs are shown for M. Profiles are shown for subject N in Figures 5.8a through 5.8f, for subject P in Figures 5.9a through 5.9f, and for subject T in Figures 5.10a through 5.10f. Average SPV from pre-flight sessions is shown as a solid line, from return sessions as a dotted line, and from recovery sessions as a dash-dot line.

5.3.4.1 Differences Within Return SPV Data

The SPV profiles for subjects N and T were analyzed for differences between CW and CCW responses, between per- and post-rotatory responses, and between head-up and dumping post-rotatory responses.

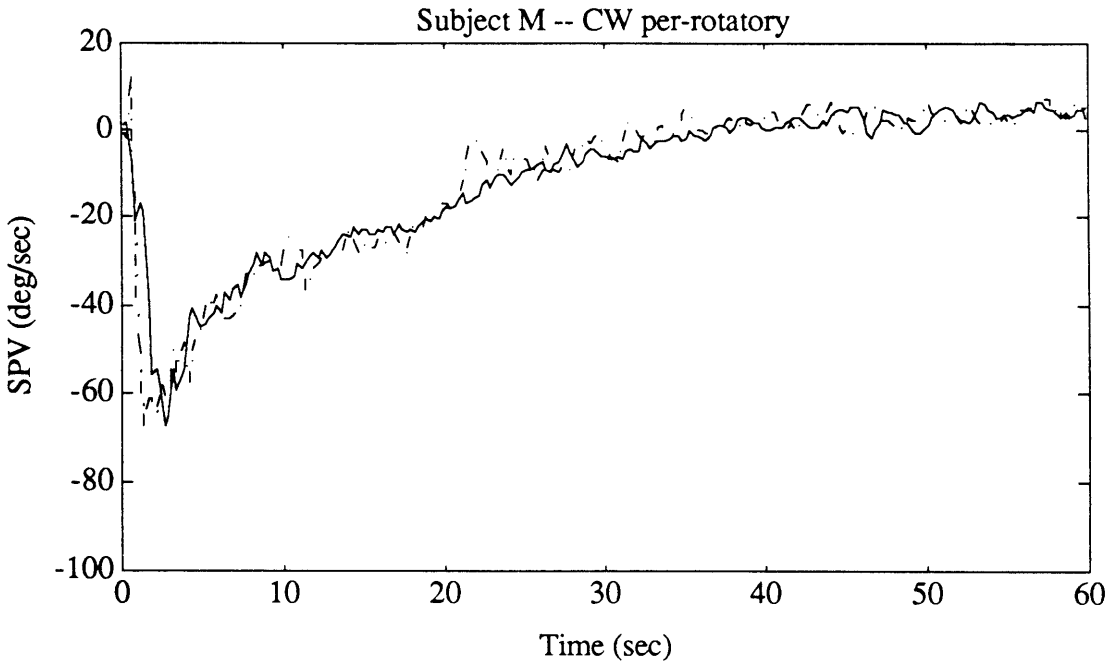


Figure 5.7a. CW per-rotatory SPV, subject M. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

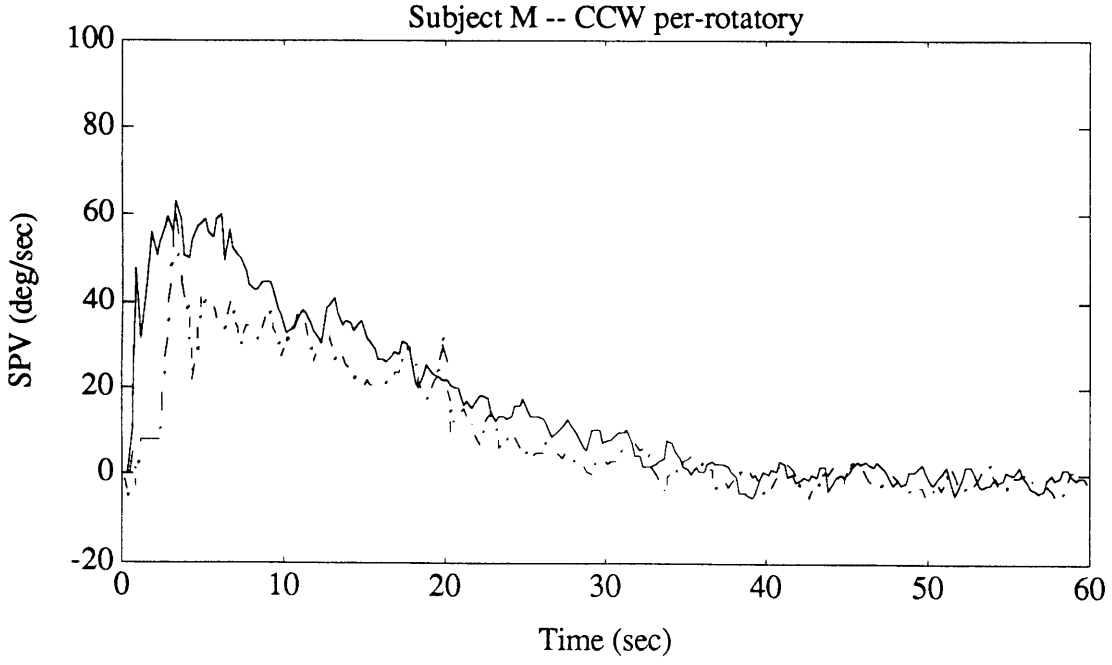


Figure 5.7b. CCW per-rotatory SPV, subject M. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

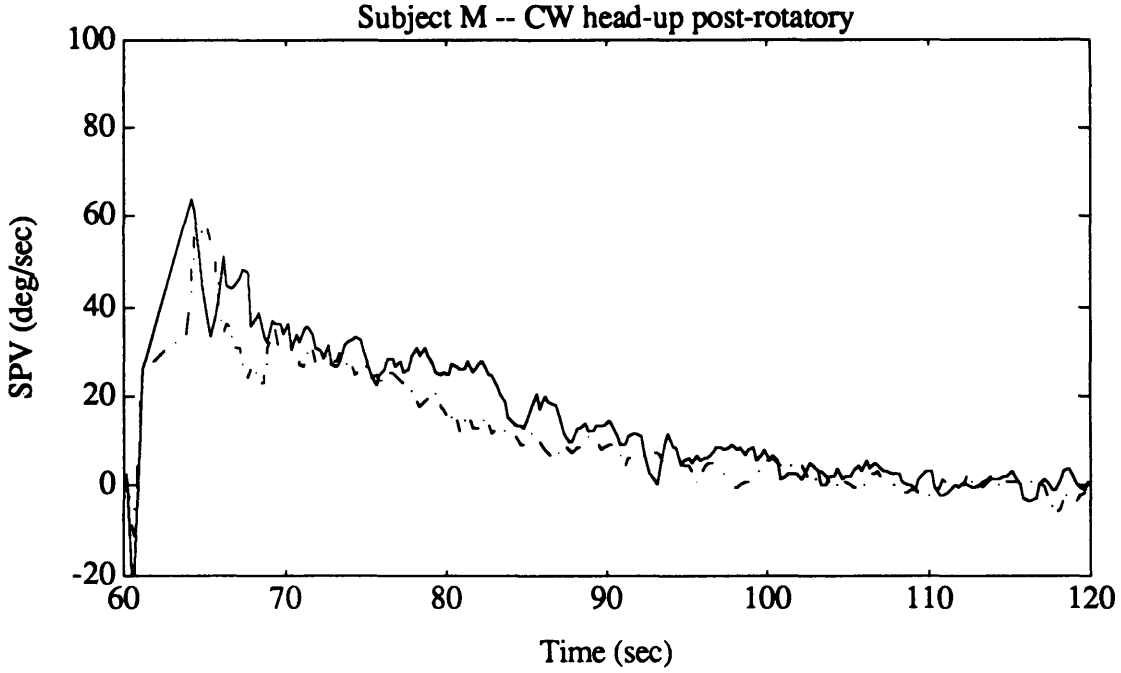


Figure 5.7c. CW head-up post-rotatory SPV, subject M. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

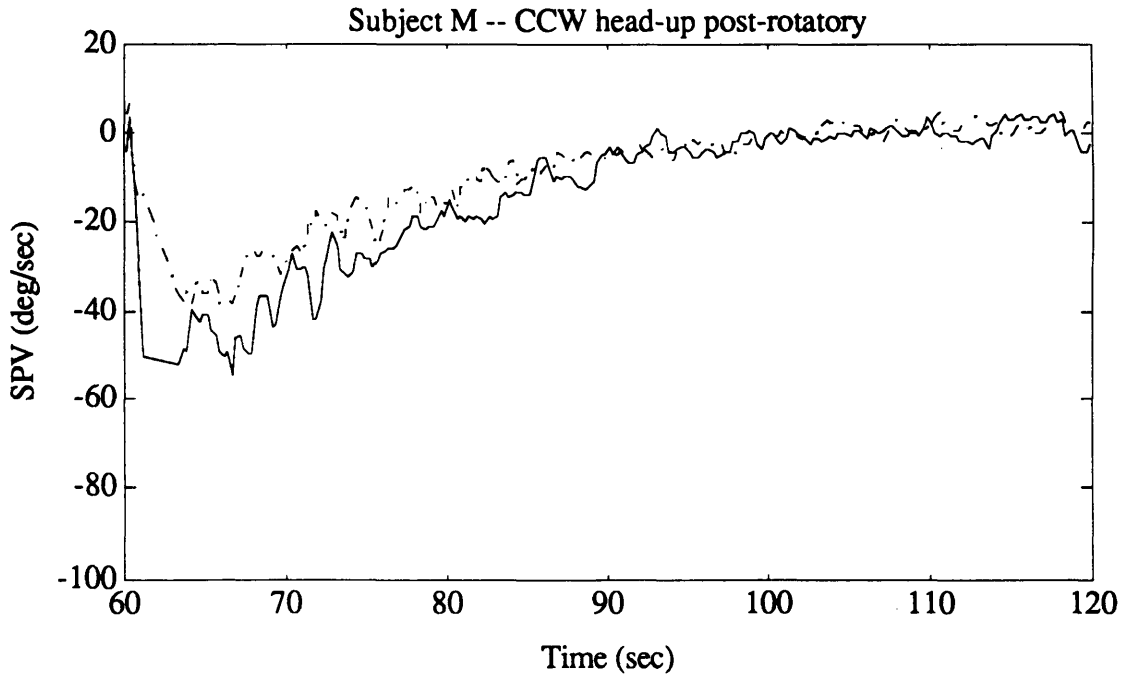


Figure 5.7d. CCW head-up post-rotatory SPV, subject M. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

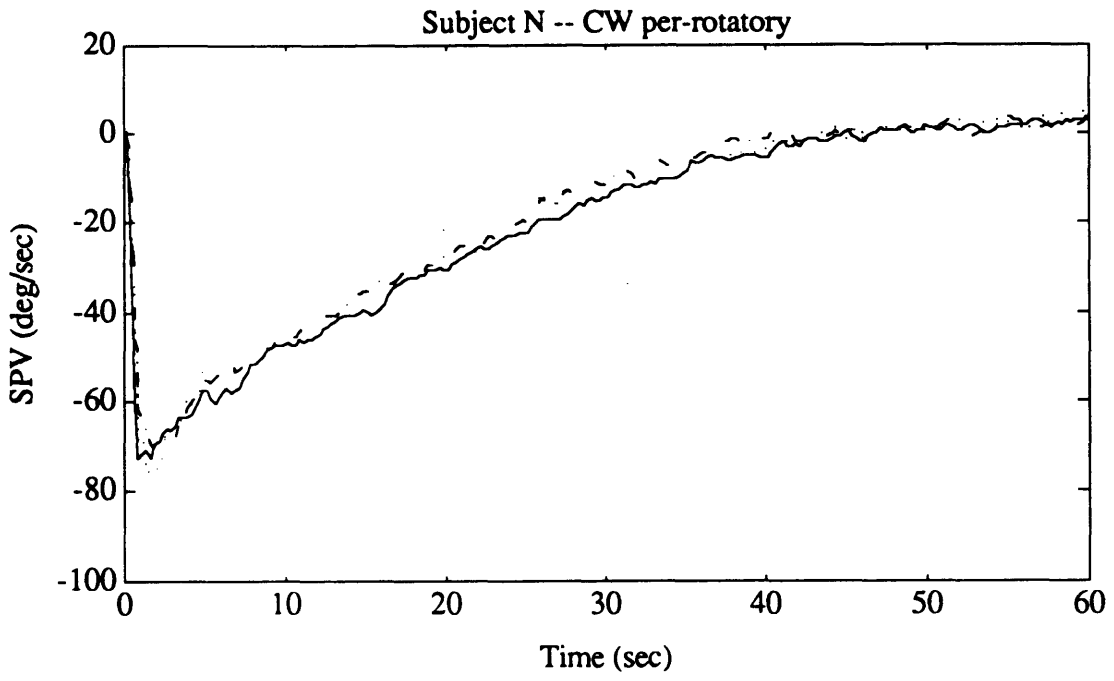


Figure 5.8a. CW per-rotatory SPV, subject N. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

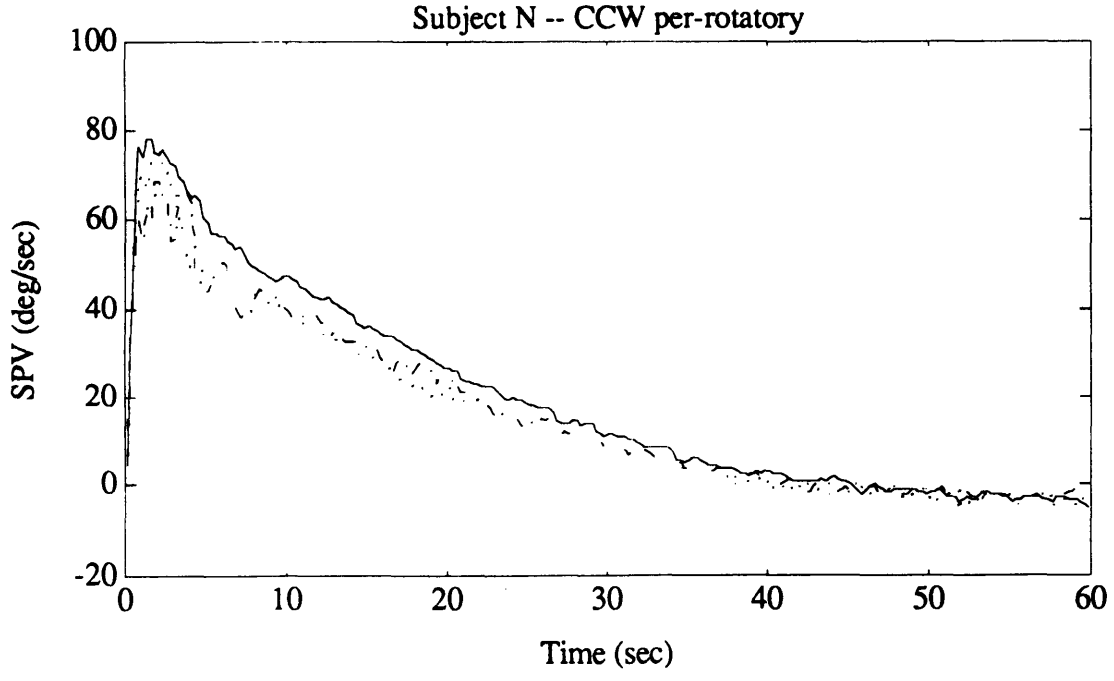


Figure 5.8b. CCW per-rotatory SPV, subject N. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

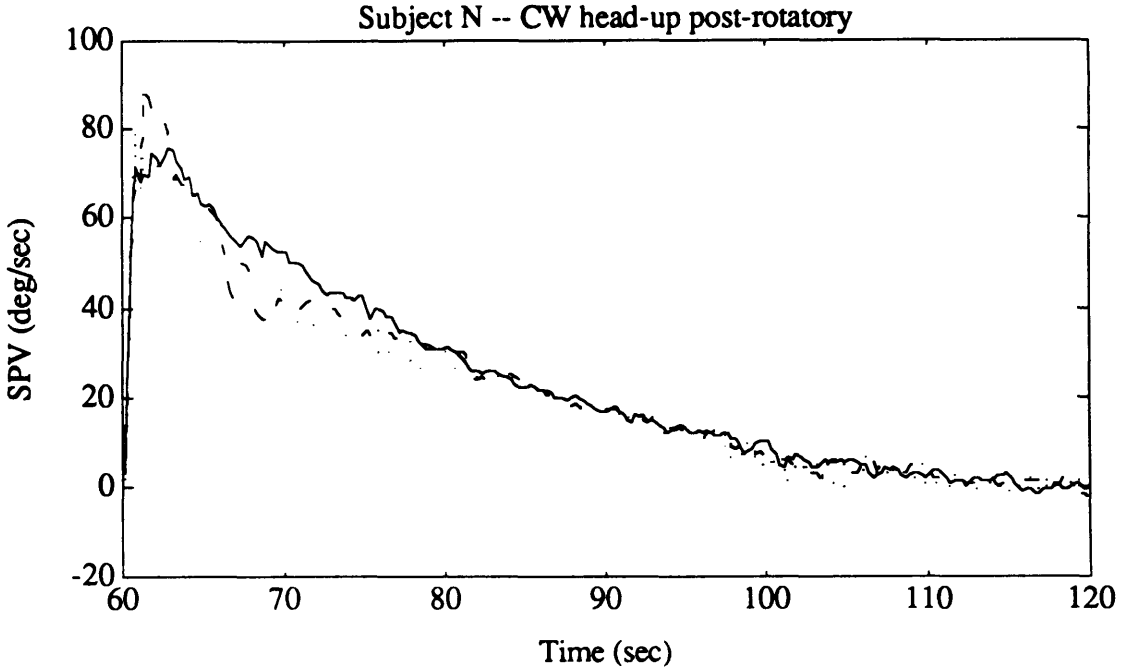


Figure 5.8c. CW head-up post-rotatory SPV, subject N. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

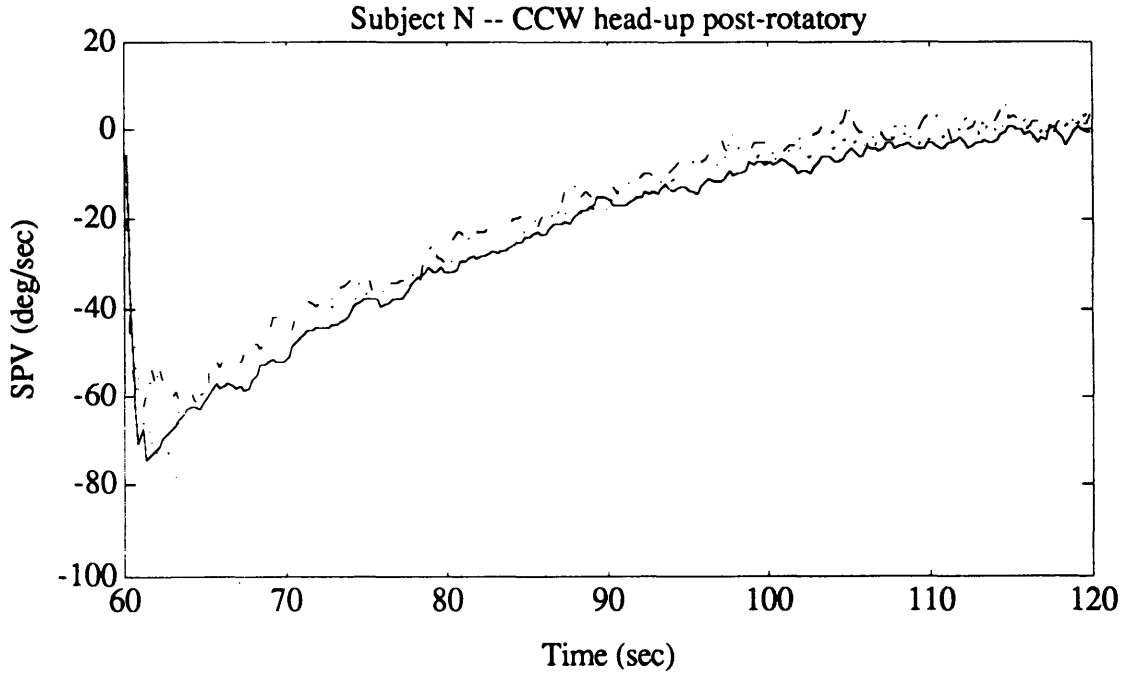


Figure 5.8d. CCW head-up post-rotatory SPV, subject N. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

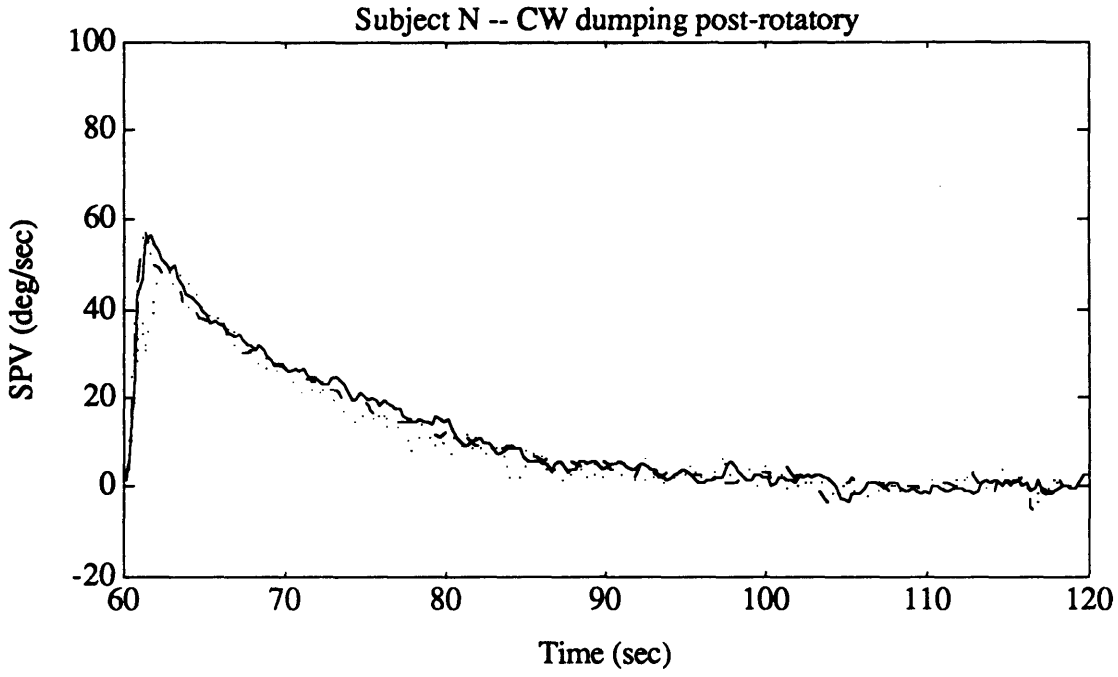


Figure 5.8e. CW dumping post-rotatory SPV, subject N. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

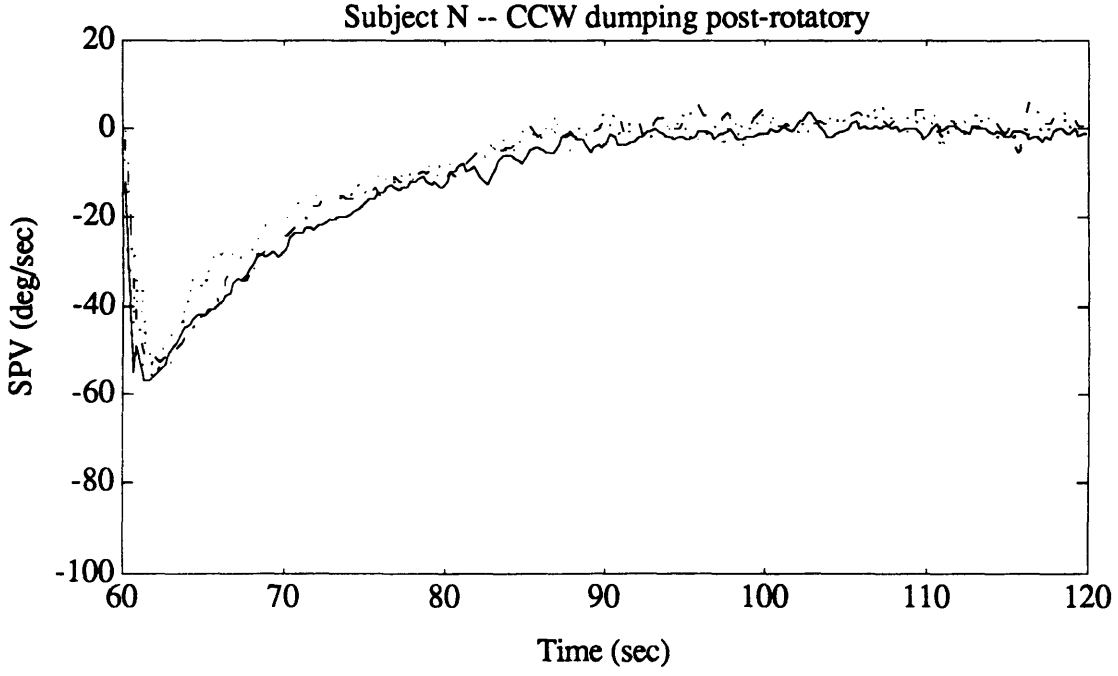


Figure 5.8f. CCW dumping post-rotatory SPV, subject N. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

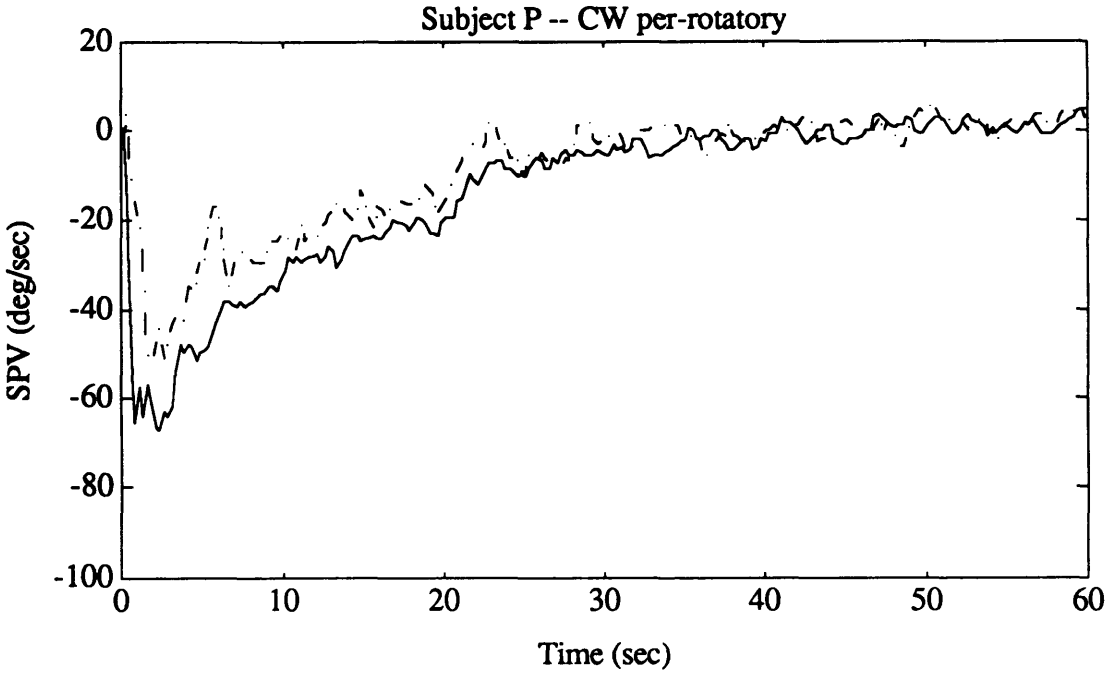


Figure 5.9a. CW per-rotatory SPV, subject P. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

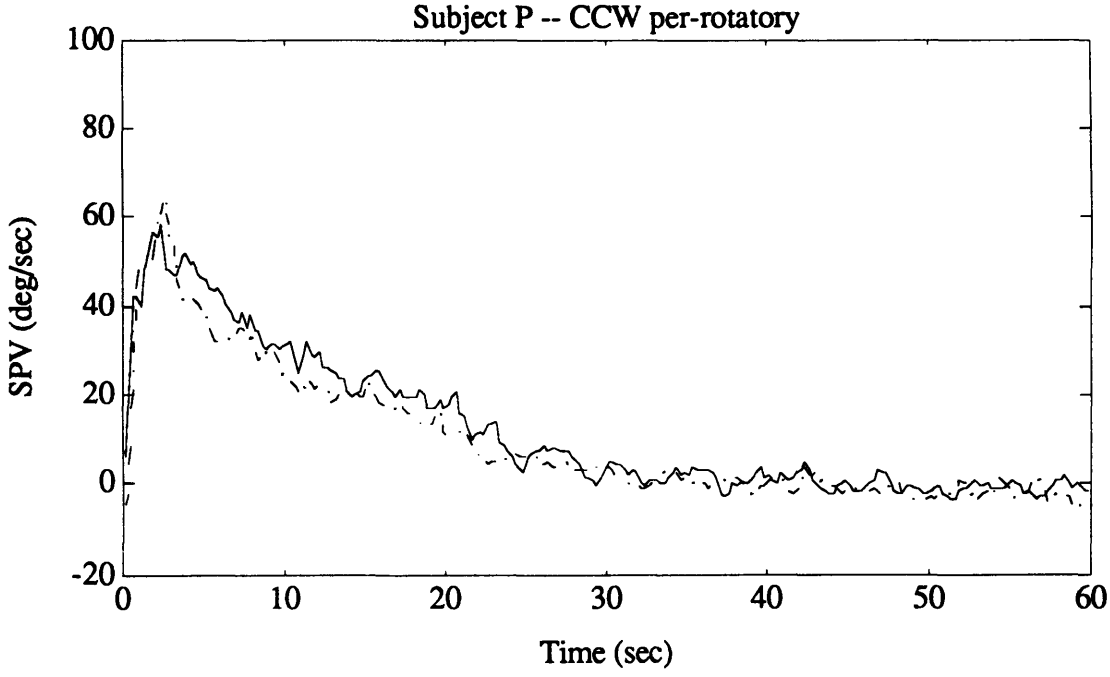


Figure 5.9b. CCW per-rotatory SPV, subject P. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

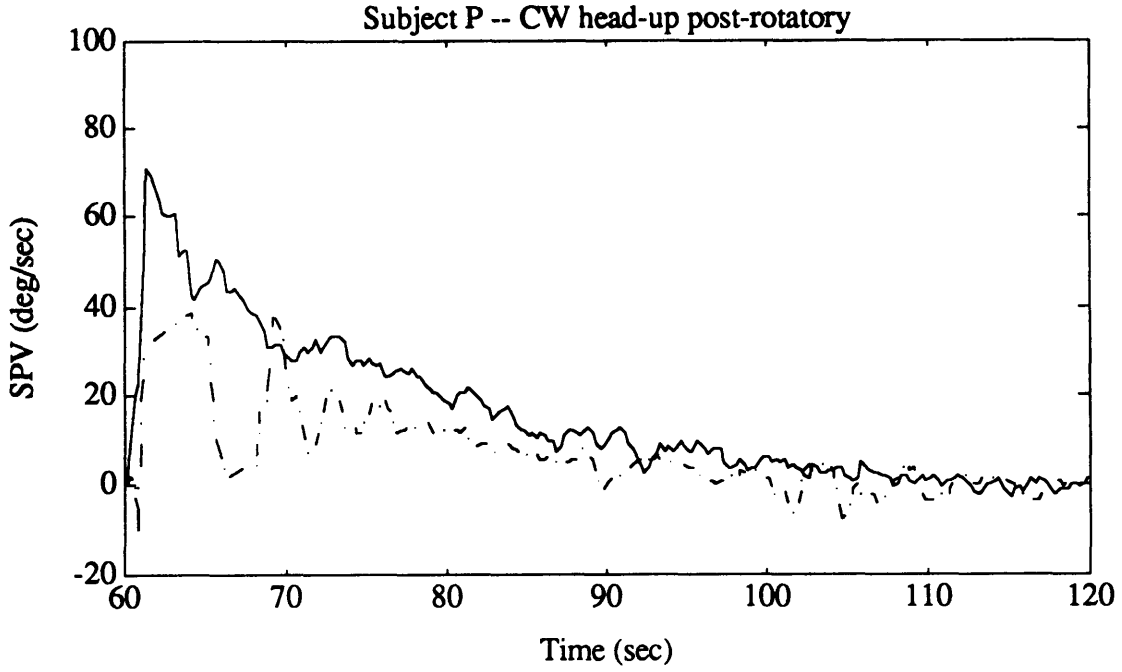


Figure 5.9c. CW head-up post-rotatory SPV, subject P. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

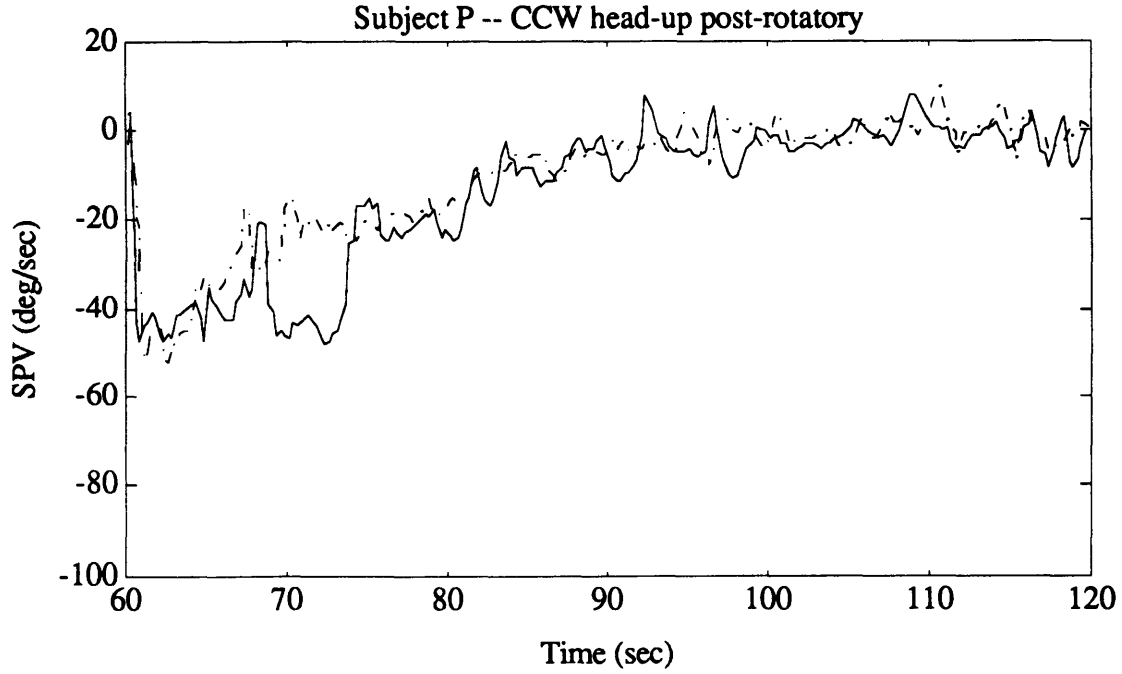


Figure 5.9d. CCW head-up post-rotatory SPV, subject P. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

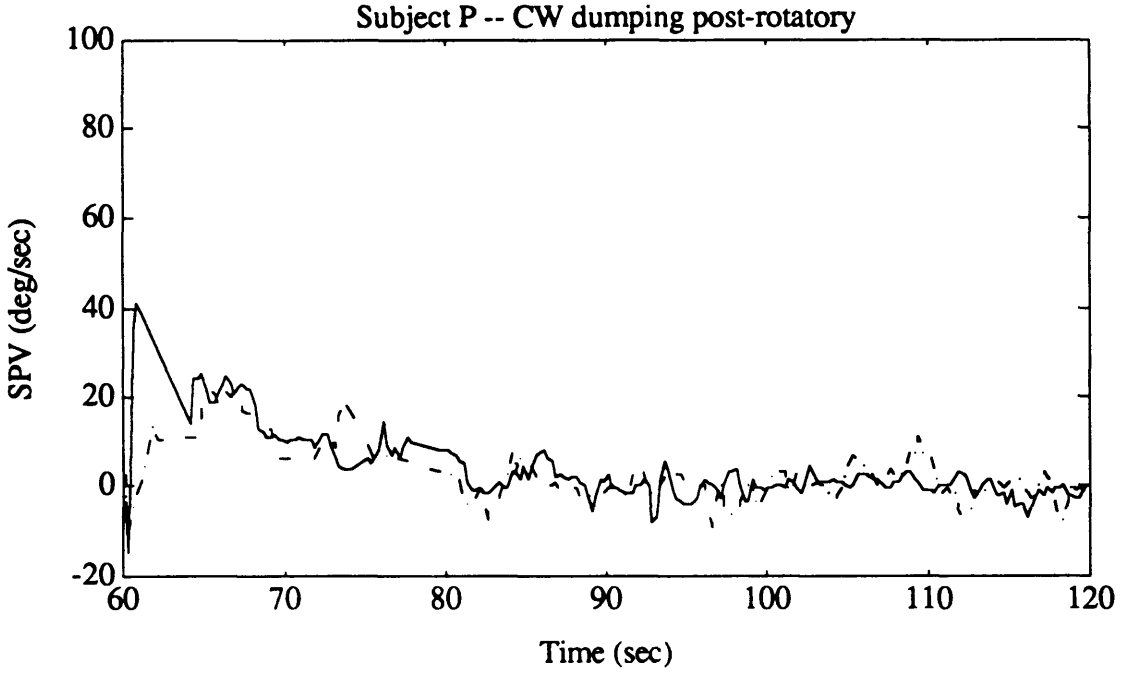


Figure 5.9e. CW dumping post-rotatory SPV, subject P. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

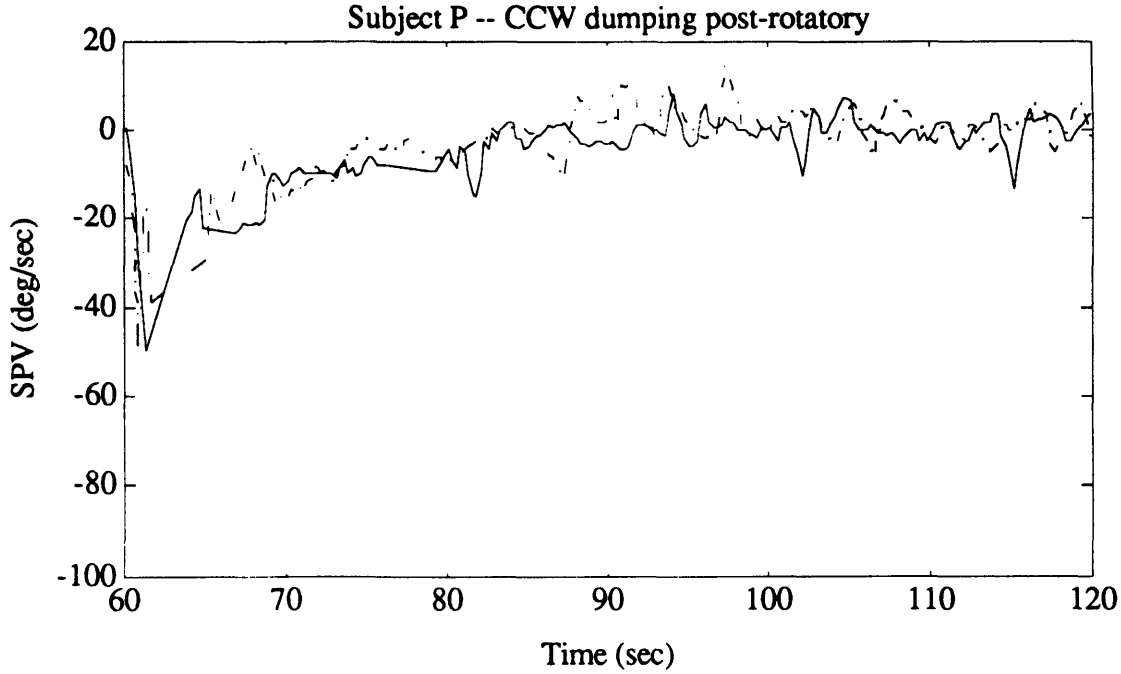


Figure 5.9f. CCW dumping post-rotatory SPV, subject P. Pre-flight (solid line) and recovery (dash-dot line) data are shown.

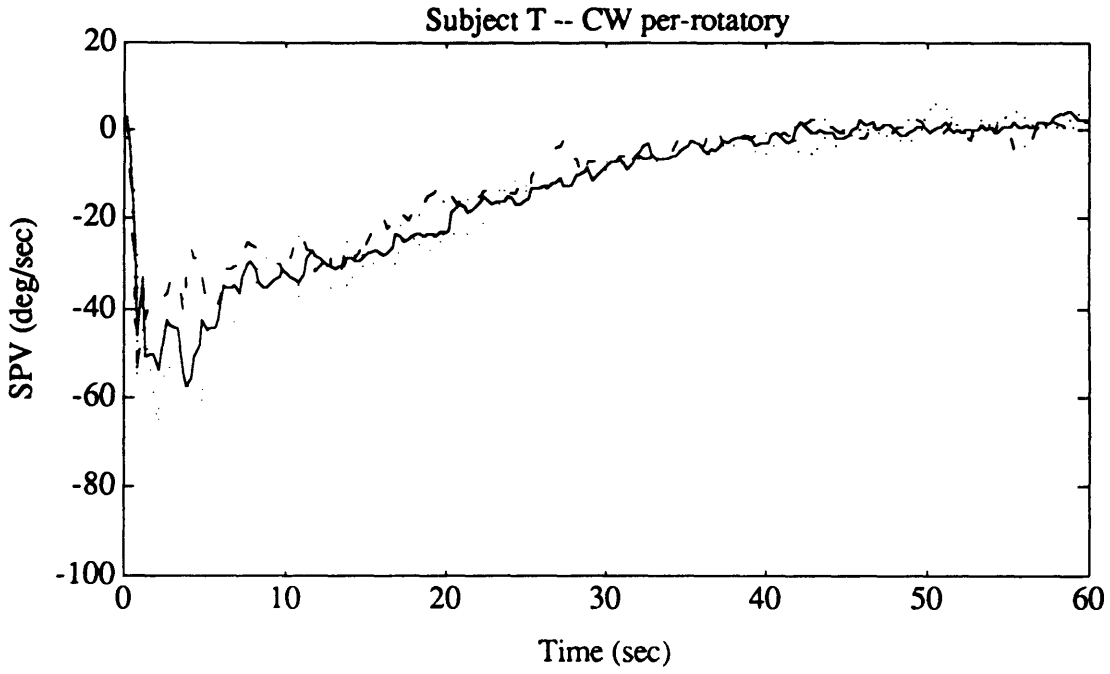


Figure 5.10a. CW per-rotatory SPV, subject T. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

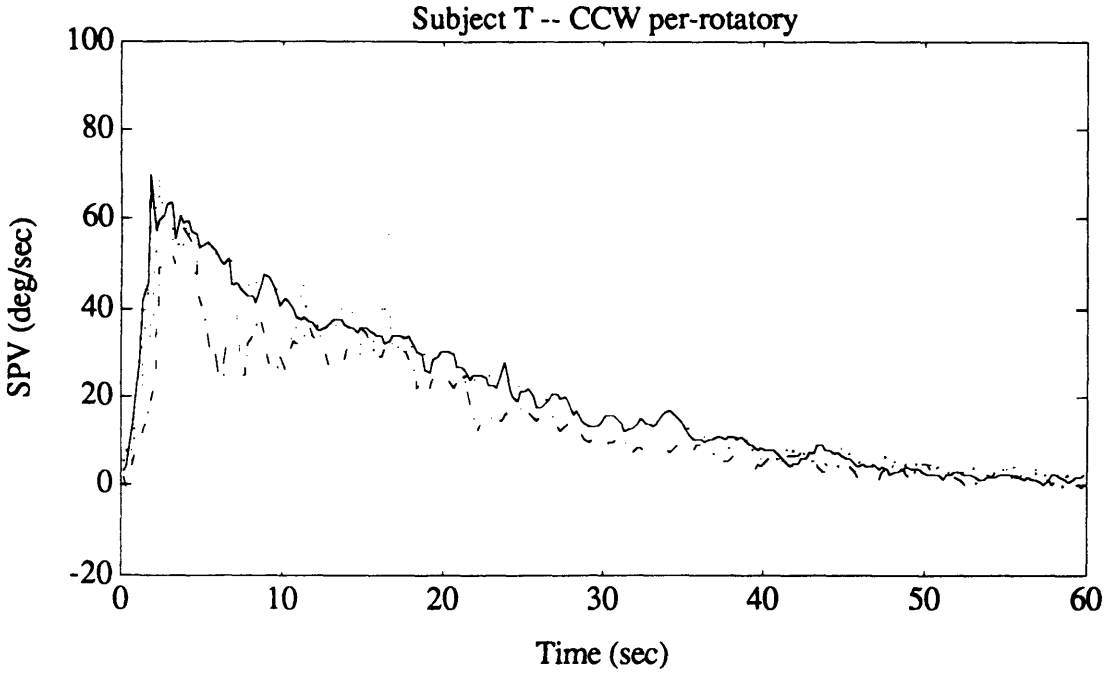


Figure 5.10b. CCW per-rotatory SPV, subject T. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

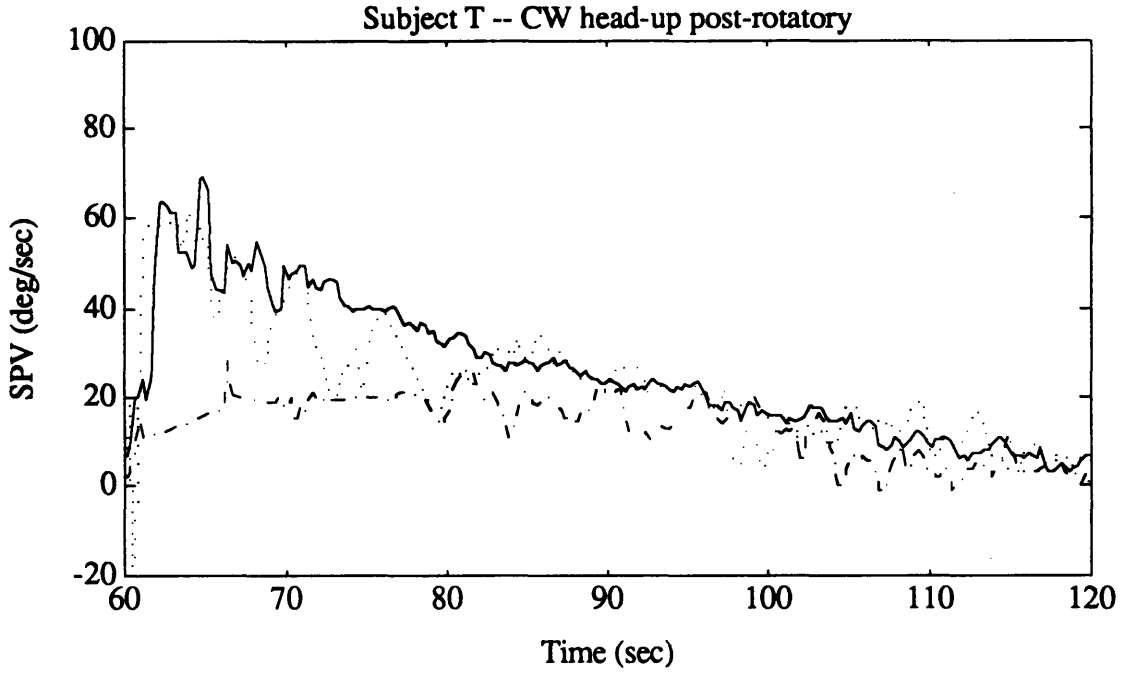


Figure 5.10c. CW head-up post-rotatory SPV, subject T. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

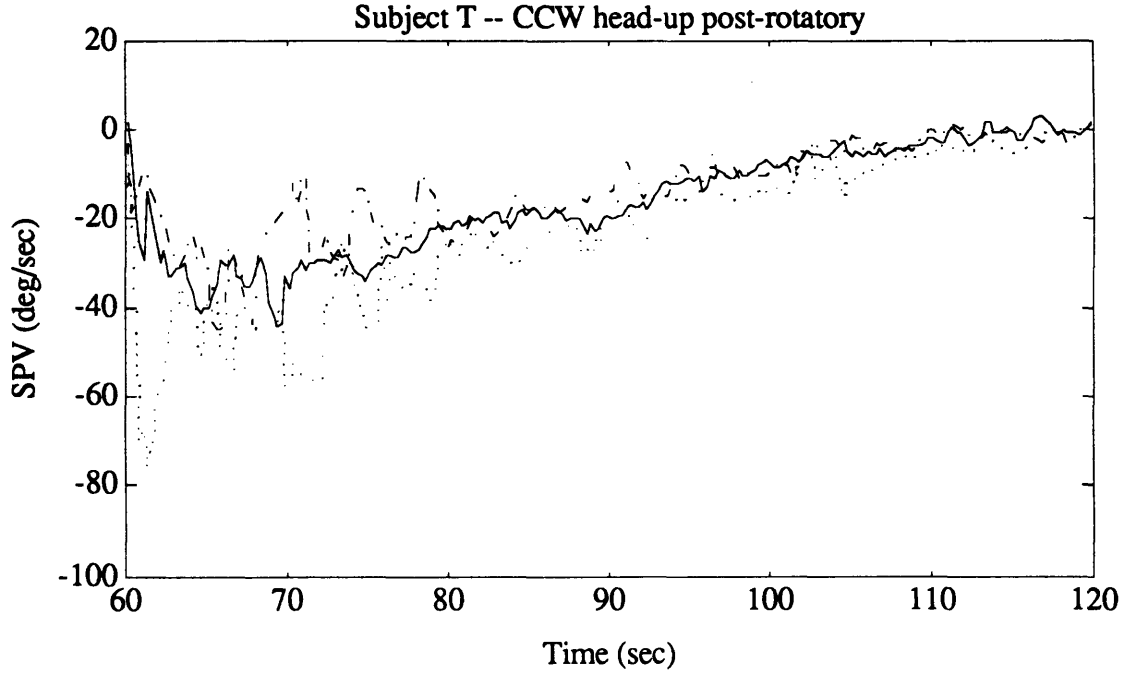


Figure 5.10d. CCW head-up post-rotatory SPV, subject T. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

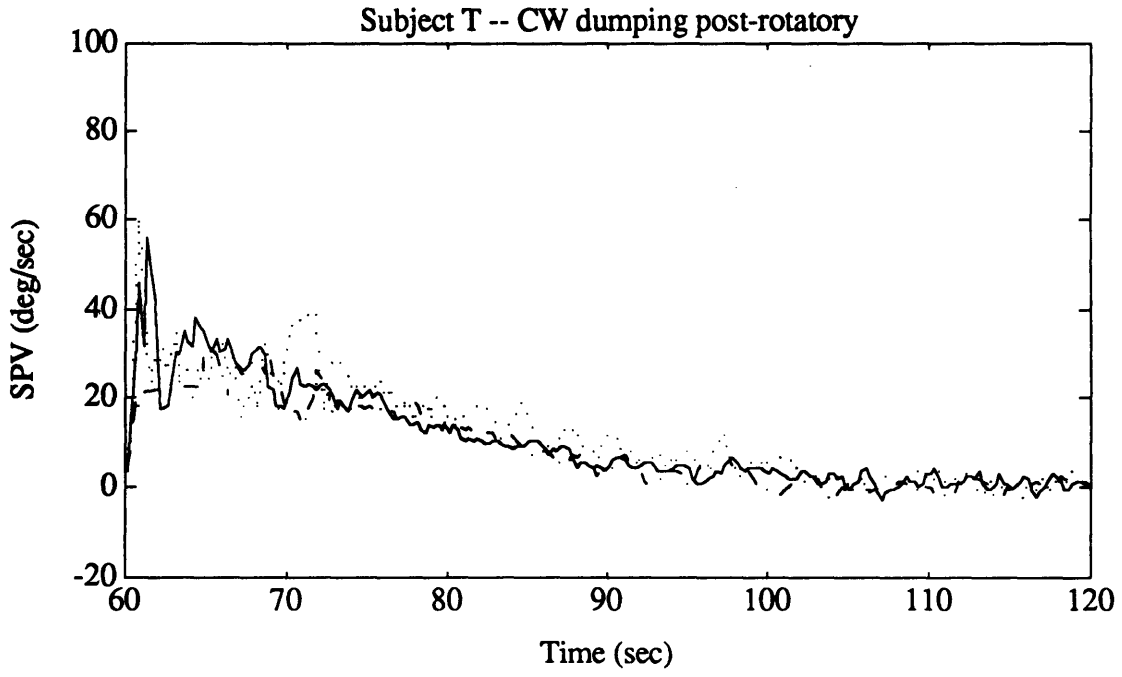


Figure 5.10e. CW dumping post-rotatory SPV, subject T. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

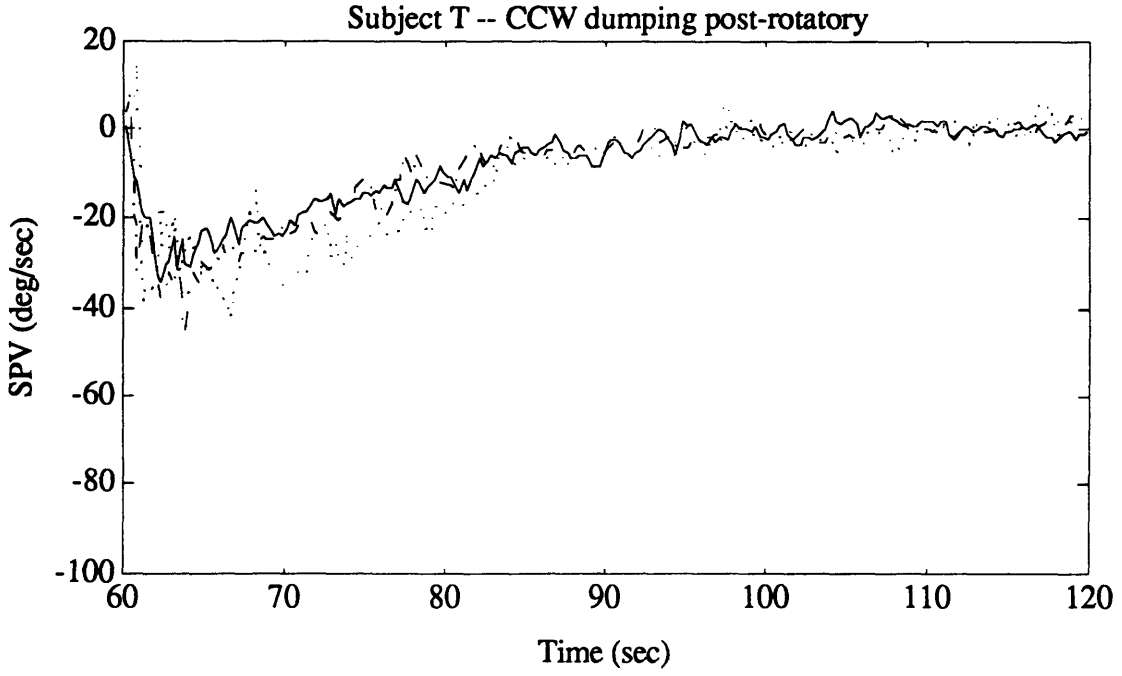


Figure 5.10f. CCW dumping post-rotatory SPV, subject T. Pre-flight (solid line), return (dotted line), and recovery (dash-dot line) data are shown.

The results for the test for directional asymmetry are summarized in Table 5.21, with the columns as described in Section 5.3.3. It was immediately notable that there was no significant difference between the CW and CCW responses for either subject, under any condition. The lack of an asymmetry in subject N was not surprising since one was not seen in the pre-flight data. However, the marked asymmetry in the pre-flight data for subject T had vanished in the return data; visual inspection of the data demonstrated that the CW responses were perhaps slightly below the CCW responses, but not by much.

The reason for the disappearance of the asymmetry is apparent in Figures 5.10a through 5.10f; the SPV response to a CCW stimulus was unchanged after exposure to weightlessness, but the SPV response to a CW stimulus (CW per-rotatory or CCW post-rotatory) was increased over the pre-flight level. The per-rotatory return data for the two directions are overlaid in Figure 5.11, to be compared with Figure 5.4b.

Subject	Time	N1	N2	dof	r	p10	p05
N	per	6.72	5.72	156	1.60	1.65	1.73
	h-up post	3.56	3.68	156	0.87	2.49	2.81
	dump post	2.69	1.92	153	2.95	7.68	7.89
T	per	6.01	6.11	152	1.45	1.65	1.72
	h-up post	2.23	2.50	114	3.20	7.09	7.69
	dump post	3.16	3.10	147	2.09	3.47	4.13

Table 5.21. Sum of t-squares tests for directional asymmetry within return data. The "Time" column refers to the portion of the SPV profiles which were compared: per- or post-rotatory, head-up or dumping. Explanation of other entries in text.

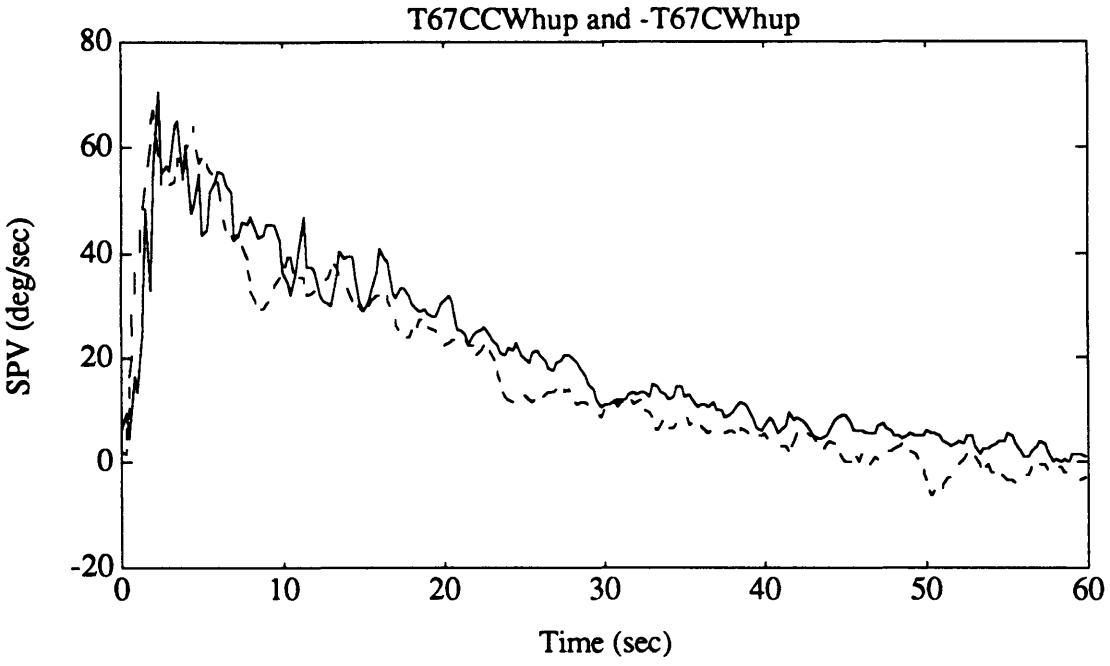


Figure 5.11. Lack of directional asymmetry in subject T, return, per-rotatory. CCW response (solid line) and negative of CW response (dashed line) are shown.

This naturally led to an inspection for directional asymmetries in the recovery data. Only subjects M and T were analyzed, since pre-flight asymmetries were present for only those subjects. The results are presented in Table 5.22. The only significant asymmetry was in the per-rotatory SPV for subject T, although the post-rotatory SPV for subject M showed a trend to significance. In general, there was a slight directional asymmetry in the recovery data, but not as great an asymmetry as was present in the pre-flight data.

Subject	Time	N1	N2	dof	r	p10	p05
M	per	4.25	5.45	148	1.12	1.82	1.96
	h-up post	5.26	5.08	145	1.82	1.78	1.89
	dump post	—	—	—	—	—	—
T	per	4.08	5.93	150	**2.87**	1.77	1.90
	h-up post	1.00	3.37	90	0.90	3.37	3.98
	dump post	3.30	3.46	138	1.80	2.93	3.40

Table 5.22. Sum of t-squares tests for directional asymmetry within recovery data. The "Time" column refers to the portion of the SPV profiles which were compared: per- or post-rotatory, head-up or dumping. Explanation of other entries in text.

The results for the test for differences between per- and post-rotatory responses are summarized in Table 5.23, with the columns as described in Section 5.3.3. In three of the four cases, there is a significant difference between the responses; in the fourth, the difference trends to significance. Both subjects exhibited similar differences pre-flight.

Subject	Direction	N1	N2	dof	r	p10	p05
N	CW	6.72	3.68	156	**4.39**	1.75	1.86
	CCW	5.72	3.56	156	**6.56**	1.86	1.99
T	CW	6.01	2.21	152	**3.56**	2.02	2.19
	CCW	6.35	2.08	130	2.03	2.00	2.17

Table 5.23. Sum of t-squares tests for differences between per- and post-rotatory responses within return data. The "Direction" column refers to the direction of the stimulus to the semi-circular canals. Explanation of other entries in text.

The results for the test for differences between head-upright and dumping post-rotatory responses are summarized in Table 5.24, with the columns as described in Section 5.3.3. In all cases, the dumping head movement leads to a significant reduction in the SPV magnitude. Both subjects exhibited similar differences pre-flight.

Subject	Direction	N1	N2	dof	r	p10	p05
N	CW	3.56	2.67	156	**27.13**	3.50	3.98
	CCW	3.67	1.92	153	**28.55**	4.34	4.85
T	CW	2.18	3.48	119	**12.21**	4.13	4.96
	CCW	2.26	3.16	143	**15.40**	4.63	5.65

Table 5.24. Sum of t-squares tests for differences between head-up and dumping post-rotatory responses within return data. The "Direction" column refers to the direction of the (per-rotatory) chair rotation. Explanation of other entries in text.

5.3.4.2 Changes from Pre-flight to Post-flight

The pre-flight SPV profiles were compared to the return and recovery SPV profiles using the same sum of t-squares tests as were used to compare profiles within the pre-flight data. Per-rotatory, head-up post-rotatory, and dumping per-rotatory responses were compared for each direction, and for each subject. The results of the comparisons between pre-flight and return data are summarized in Table 5.25; notice that comparisons were only made for subjects N and T, since there were far too many dropouts in the return data from other subjects to make it analyzable. The results of the comparisons between pre-flight and recovery data are summarized in Table 5.26.

For subject N, there were significant decreases from pre-flight to return in the per-rotatory SPV (CW and CCW). The other stimuli produced SPV profiles which were visibly, but not significantly, reduced. The recovery data lay, in general, between the pre-flight and return levels; however, the CCW head-up post-rotatory SPV was lower than both, and significantly lower than the pre-flight SPV. The recovery per-rotatory SPV was significantly reduced from the pre-flight levels in both directions, but the difference was less than that between the pre-flight and return data.

For subject T, the CCW dumping post-rotatory SPV response was significantly increased from pre-flight levels, while the other CW stimuli (CW per-rotatory and CCW head-up post-rotatory) produced SPV profiles which were visibly, but not significantly, increased. The SPV for subject T resulting from CCW stimuli was unchanged from the pre-flight levels. These changes were reflected in the disappearance of the pre-flight directional asymmetry.

In the recovery sessions for subject T, the SPV evoked by a CCW stimuli was generally lower than the pre-flight levels; the decrease in CCW per-rotatory SPV was significant and the decrease in the CW head-up post-rotatory SPV trended to significance. The SPV evoked by CW stimuli was equivalent to the pre-flight levels, reduced from the return days. This resulted in a slight directional asymmetry, with but to a lesser extent than pre-flight.

For subject M, the SPV was generally decreased in the recovery sessions from pre-flight levels. The CCW per-rotatory and post-rotatory responses were significantly lower, while the CW post-rotatory difference trended towards significance. The CW per-rotatory SPV was not significantly different.

For subject P, the SPV was significantly decreased in the recovery sessions from pre-flight levels for the CW per-rotatory and head-up post-rotatory responses. The SPV was slightly, but not significantly lower for the CCW per-rotatory and head-up post-rotatory responses, while there was no change in the dumping responses.

Subj	Dir	Time	N1	N2	dof	r	p10	p05
N	CW	per	15.47	6.72	156	**4.10**	1.40	1.46
		h-up post	6.78	3.56	156	1.65	1.76	1.86
		dump post	7.25	2.67	156	1.02	1.79	1.90
	CCW	per	15.42	5.72	156	**4.04**	1.43	1.49
		h-up post	7.29	3.68	156	1.09	1.71	1.82
		dump post	6.76	1.92	153	1.94	1.98	2.14
T	CW	per	9.25	5.97	153	1.33	1.59	1.64
		h-up post	6.47	2.08	130	1.12	2.13	2.15
		dump post	5.07	3.07	153	1.73	1.99	2.25
	CCW	per	10.46	5.97	156	0.34	1.51	1.60
		h-up post	5.61	2.24	148	1.67	1.97	2.40
		dump post	3.88	3.09	148	**3.84**	2.37	2.80

Table 5.25. Sum of t-squares tests for differences between pre-flight and return SPV data. The "Dir" column refers to the direction of the (per-rotatory) chair rotation. The "Time" column refers to the portion of the SPV profiles which were compared: per- or post-rotatory, head-up or dumping. Explanation of other entries in text.

Subj	Dir	Time	N1	N2	dof	r	p10	p05
M	CW	per	8.96	4.12	156	0.87	1.60	1.67
		h-up post	4.00	5.34	143	1.91	1.85	2.00
		dump post	—	—	—	—	—	—
	CCW	per	6.07	5.39	150	**2.68**	1.69	1.77
		h-up post	4.96	5.05	146	**2.48**	1.80	1.93
		dump post	—	—	—	—	—	—
N	CW	per	15.47	7.56	156	**1.61**	1.38	1.44
		h-up post	6.78	3.51	156	0.71	1.76	1.87
		dump post	7.25	3.83	156	0.45	1.71	1.82
	CCW	per	15.42	6.28	156	**2.55**	1.42	1.48
		h-up post	7.29	3.42	156	**2.14**	1.72	1.82
		dump post	6.71	2.81	156	1.79	1.83	1.94
P	CW	per	8.20	2.93	150	**2.57**	1.67	1.77
		h-up post	6.13	2.32	134	**3.51**	1.99	2.14
		dump post	2.67	1.77	111	1.26	7.88	8.14
	CCW	per	8.04	5.61	156	1.21	1.59	1.69
		h-up post	2.47	3.17	144	2.12	4.33	5.26
		dump post	1.91	1.89	95	2.01	3.62	4.23
T	CW	per	9.13	3.96	156	1.10	1.59	1.67
		h-up post	7.16	1.00	104	2.16	2.13	2.29
		dump post	5.17	3.31	144	0.73	1.99	2.16
	CCW	per	10.60	5.86	152	**1.90**	1.51	1.59
		h-up post	5.77	2.77	141	1.23	1.97	2.12
		dump post	3.86	3.34	146	1.30	2.37	2.68

Table 5.26. Sum of t-squares tests for differences between pre-flight and recovery SPV data. The "Dir" column refers to the direction of the (per-rotatory) chair rotation. The "Time" column refers to the portion of the SPV profiles which were compared: per- or post-rotatory, head-up or dumping. Explanation of other entries in text.

5.4 Model Fitting

The three-parameter model described in Section 4.9 was fit to the average SPV profiles for all four subjects, and to the individual SPV profiles for subject N. In general, the other subjects' responses contained so many dropouts that there was insufficient information near the peak SPV for reliable model fits to individual runs. Only those average responses which were "analyzable", as described in Section 5.3.4, were fit with models.

As usual, the data were divided into "pre-flight", "return", and "recovery" sessions; within those divisions the data were combined to determine per-rotatory, head-upright post-rotatory, and dumping post-rotatory average responses.

5.4.1 Fits to Individual Runs

The optimal model parameters for each individual run for Subject N are shown in Tables 5.27 through 5.29. Tables 5.27a and 5.27b contain the overall system gain, K, for the per- and post-rotatory sections respectively. Similarly, Tables 5.28a and 5.28b contain the indirect pathway gain, g_0, and Tables 5.29a and 5.29b contain the velocity storage time constant, $1/h_0$.

Although the limits on the parameters are quite wide, there were four runs (N405 and N1008 per-rotatory, N507 and N708 post-rotatory) in which g_0 reached the upper limit

of 0.45, artificially driving down the system gain K. In all of these cases, the SPV increased to its peak value more slowly than usual, thereby producing a SPV which was lower than normal for the first 3-5 seconds; or, there were early dropouts so that there was little data available about the peak. The corresponding model parameters were discarded. Figures 5.12a and 5.12b show examples of a good model fit (N209), and one of the runs in which g_0 reached the upper limit (N708).

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2	0.590	0.647	0.539	0.521	0.720	0.644	0.498	0.599
3	0.633	0.600	0.422	0.633	0.548	0.689	0.288	0.608
4	0.556	0.590	0.458	0.516	0.606	0.574	0.549	0.607
5	0.776	0.792	*0.203*	0.623	0.473	0.622	0.366	0.599
7	0.794	0.467	0.571	0.370	0.491	0.330	0.670	0.543
8	0.717	0.488	0.794	0.689	0.619	0.637	0.440	*0.179*
9	0.755	0.805	0.796	0.552	0.771	—	0.753	0.410
10	0.873	0.688	0.665	0.744	—	—	—	0.684

Table 5.27a. System gain, K, from optimal three-parameter model fits to per-rotatory portions of individual runs for subject N. Underlined parameters are discarded. – indicates missing data.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2	0.604	0.612	0.506	0.555	0.575	0.630	0.514	0.516
3	0.515	0.509	0.498	0.660	0.609	0.419	0.379	0.514
4	0.414	0.519	0.377	0.417	0.512	0.421	0.400	0.403
5	—	0.695	0.425	0.517	0.521	0.433	0.520	0.432
7	0.706	0.480	0.552	*0.141*	0.413	0.621	0.640	0.737
8	0.663	0.531	0.596	0.402	*0.187*	0.591	0.386	0.529
9	0.479	0.628	0.516	0.463	0.590	—	0.527	0.388
10	0.446	0.469	0.508	0.526	—	—	—	0.566

Table 5.27b. System gain, K, from optimal three-parameter model fits to post-rotatory portions of individual runs for subject N. Underlined parameters are discarded. – indicates missing data.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2	0.164	0.160	0.154	0.152	0.090	0.099	0.174	0.151
3	0.127	0.124	0.171	0.117	0.120	0.084	0.240	0.122
4	0.149	0.186	0.171	0.149	0.108	0.110	0.125	0.125
5	0.114	0.143	*0.450*	0.107	0.137	0.098	0.174	0.140
7	0.134	0.135	0.162	0.231	0.183	0.282	0.131	0.190
8	0.135	0.118	0.118	0.106	0.119	0.123	0.175	*0.450*
9	0.125	0.133	0.109	0.144	0.091	—	0.100	0.213
10	0.098	0.130	0.113	0.089	—	—	—	0.113

Table 5.28a. Indirect pathway gain, g_0, from optimal three-parameter model fits to per-rotatory portions of individual runs for subject N. Underlined parameters are discarded. – indicates missing data.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2	0.140	0.141	0.139	0.094	0.105	0.083	0.143	0.143
3	0.166	0.154	0.113	0.089	0.092	0.180	0.166	0.108
4	0.106	0.082	0.068	0.093	0.056	0.080	0.086	0.100
5	—	0.044	0.080	0.048	0.028	0.088	0.070	0.128
7	0.160	0.161	0.154	*0.450*	0.163	0.115	0.106	0.072
8	0.161	0.117	0.159	0.193	*0.450*	0.147	0.281	0.146
9	0.113	0.045	0.084	0.059	0.043	—	0.061	0.116
10	0.097	0.073	0.080	0.049	—	—	—	0.056

Table 5.28b. Indirect pathway gain, g_0, from optimal three-parameter model fits to post-rotatory portions of individual runs for subject N. Underlined parameters are discarded. – indicates missing data.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2	29.49	38.52	33.88	31.42	38.92	26.89	28.48	30.25
3	32.06	29.97	25.92	24.80	24.45	35.48	23.81	32.34
4	25.53	29.96	23.18	25.47	27.22	26.60	29.02	28.88
5	31.97	29.13	21.16	31.98	21.11	33.38	22.24	28.81
7	33.55	31.94	28.93	25.85	27.02	24.00	26.28	28.94
8	32.00	29.07	22.76	25.26	21.41	23.96	19.08	22.09
9	29.91	28.36	26.70	27.53	28.92	—	21.63	20.42
10	30.99	28.81	27.89	30.10	—	—	—	24.99

Table 5.29a. Velocity storage time constant, $1/h_0$, from optimal three-parameter model fits to per-rotatory portions of individual runs for subject N. Underlined parameters are discarded. – indicates missing data.

Run	BDC2	BDC3	BDC4	BDC5	BDC7	BDC8	BDC9	BDC10
2	23.70	24.37	26.87	27.98	26.16	31.02	27.00	29.16
3	21.62	27.68	26.65	24.37	25.50	19.70	23.59	17.62
4	22.68	13.34	21.25	12.16	18.91	19.90	18.82	15.85
5	—	32.63	14.77	16.36	33.99	8.40	6.48	9.00
7	20.72	25.21	22.83	20.10	18.64	27.78	20.41	32.20
8	22.63	27.41	22.81	22.17	20.70	20.74	16.63	17.70
9	22.24	17.81	18.07	18.98	15.10	—	19.40	17.29
10	14.45	16.59	13.26	17.96	—	—	—	13.07

Table 5.29b. Velocity storage time constant, $1/h_0$, from optimal three-parameter model fits to post-rotatory portions of individual runs for subject N. Underlined parameters are discarded. – indicates missing data.

In several cases, one or more of the parameters were significantly higher or lower than the parameters for similar runs, deviating from the mean parameter by more than three standard deviations. In any given set of stimuli (e.g. pre-flight CW head-up post-rotatory), a run was discarded if one or more of its model parameters was further than three standard deviations from the mean.

The parameters which were discarded are underlined in the tables. The four runs in which g_0 reached the upper limit have also been italicized.

The mean and standard deviation for the parameters for each stimulus type are shown in Tables 5.30 and 5.31 respectively. Due to the high variability in the parameters, and the low number of data points, there were very few significant conclusions which could be drawn from the parametric fits. However, some qualitative differences were easily visible.

In general, there were no directional asymmetries in the parameters. There was a statistically significant asymmetry in g_0 during the pre-flight per-rotatory runs, and in h_0 during the recovery dumping post-rotatory runs, as determined by standard t-tests. However, these differences were not consistent and were rejected as chance occurrences.

Both K and $1/h_0$ were consistently smaller for the post-rotatory head-up data than for the per-rotatory. Even though only three of the differences were significant (CCW pre-flight K, CW and CCW pre-flight h_0), the consistency suggested that the differences were real. Paired t-tests showed significant decreases in K ($t = 3.14$, $df = 5$, $p < 0.05$) and $1/h_0$ ($t = 2.84$, $df = 5$, $p < 0.05$). There was no significant or consistent change in g_0.

The most significant differences were decreases in K, g_0, and $1/h_0$ for the dumping runs, as compared to the head-upright post-rotatory runs. Overall, 10 of the 18 differences were statistically significant at the $p < 0.05$ level.

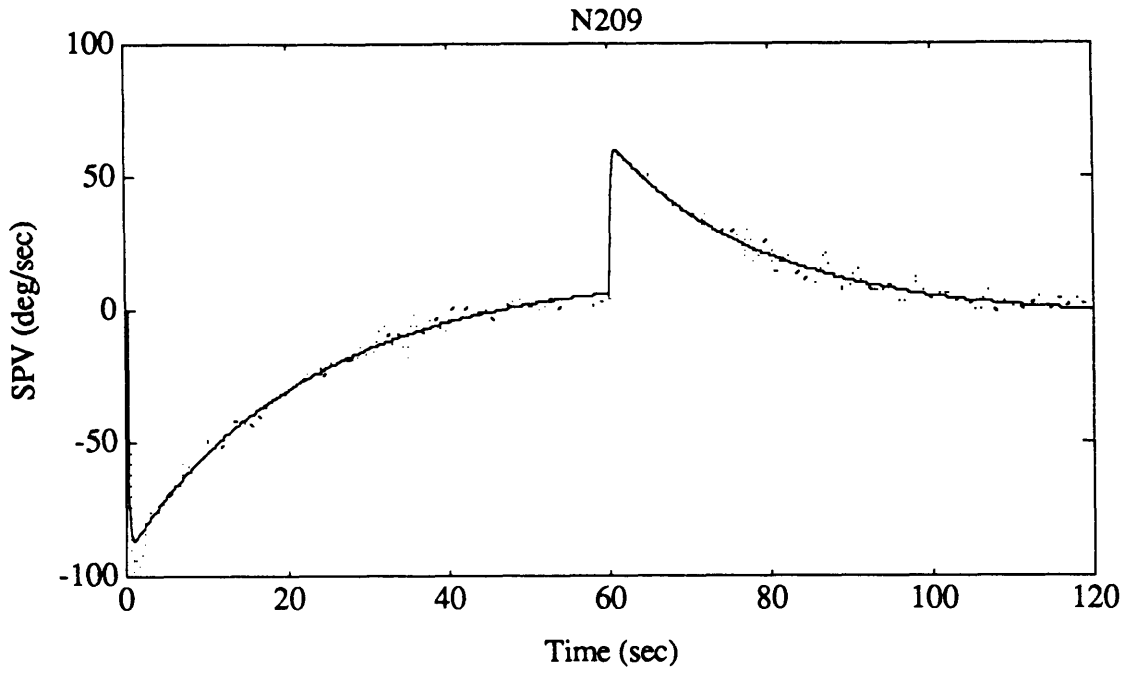


Figure 5.12a. Example of a good model fit to an individual run (N209). The SPV data is shown as the dotted line, and the model fit as the solid line.

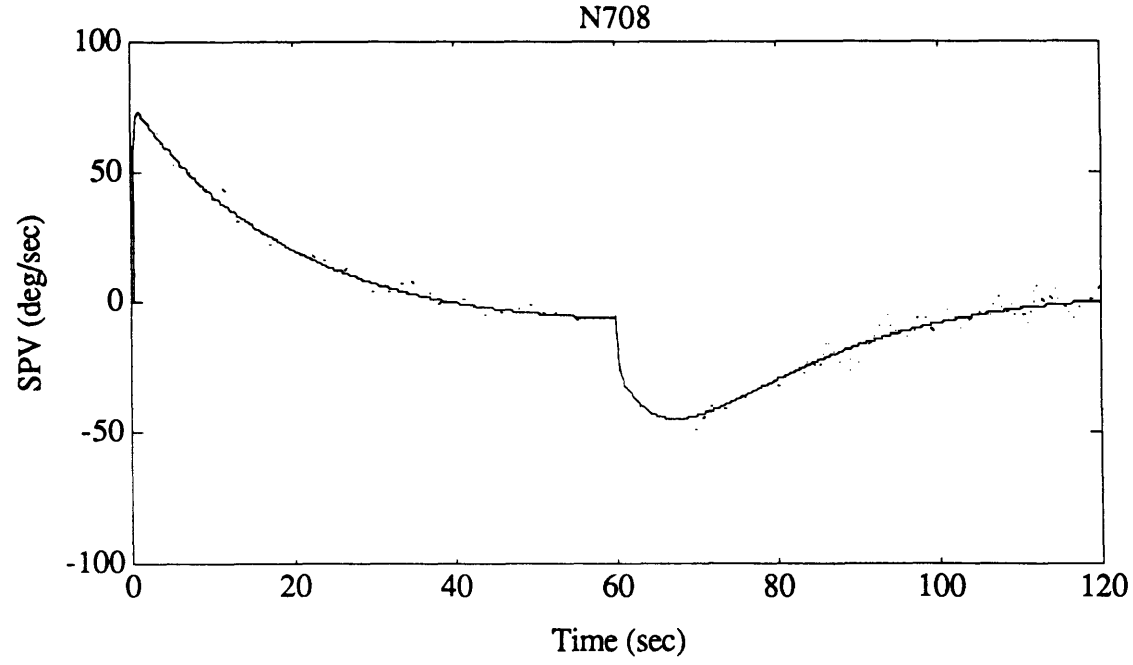


Figure 5.12b. Example of a bad model fit to an individual run (N708). The SPV data is shown as the dotted line, and the model fit as the solid line. The indirect pathway gain for this run reached the maximum of 0.45.

Sessions	Direction	Time	K	g_0	$1/h_0$	n
pre	CW	per	0.610	0.148	29.62	15
		h-up post	0.574	0.141	24.53	7
		dump post	0.477	0.081	18.32	8
	CCW	per	0.676	0.121	28.85	15
		h-up post	0.547	0.144	24.42	8
		dump post	0.482	0.071	15.57	6
return	CW	per	0.634	0.114	29.26	6
		h-up post	0.560	0.117	25.90	4
		dump post	0.508	0.060	17.97	3
	CCW	per	0.598	0.114	26.63	6
		h-up post	0.540	0.140	21.98	3
		dump post	0.477	0.058	21.20	2
recovery	CW	per	0.603	0.142	27.64	7
		h-up post	0.602	0.116	27.19	4
		dump post	0.430	0.091	17.84	4
	CCW	per	0.583	0.138	26.31	4
		h-up post	0.474	0.140	19.64	3
		dump post	0.506	0.085	9.52	3

Table 5.30. Mean values for the three-parameter model fits to SPV data for subject N.

Sessions	Direction	Time	K	g_0	$1/h_0$	n
pre	CW	per	0.120	0.019	3.89	15
		h-up post	0.075	0.023	2.45	7
		dump post	0.079	0.023	3.90	8
	CCW	per	0.118	0.020	2.97	15
		h-up post	0.088	0.034	2.49	8
		dump post	0.041	0.019	1.71	6
return	CW	per	0.101	0.035	4.80	6
		h-up post	0.101	0.034	5.24	4
		dump post	0.085	0.019	2.53	3
	CCW	per	0.076	0.019	6.22	6
		h-up post	0.105	0.044	3.09	3
		dump post	0.062	0.042	18.09	2
recovery	CW	per	0.086	0.031	2.90	7
		h-up post	0.108	0.034	5.00	4
		dump post	0.065	0.023	1.60	4
	CCW	per	0.103	0.027	5.68	4
		h-up post	0.083	0.029	3.42	3
		dump post	0.068	0.038	3.33	3

Table 5.31. Standard deviations for three-parameter model fits to SPV data for N.

Of primary interest were the differences between the pre-flight and post-flight parameters. These changes are shown in bar graph form in Figure 5.13. As described in Section 5.3, the SPV data for subject N was generally reduced from the pre-flight baseline during the return sessions, but was not significantly changed during the recovery sessions. There were no significant changes in K from pre-flight to post-flight for any of the stimulus types. The only significant change in $1/h_0$ was a decrease for the CCW dumping post-rotatory data from pre-flight to recovery; since this was the only difference, and it did not show up during the return sessions, it was rejected as a chance occurrence. However, g_0 was reduced from pre-flight to return sessions for all six conditions. While only the CW per-rotatory g_0 was significantly reduced, this consistency correlated well with the reduction in SPV. A paired t-test demonstrated a significant decrease ($p < 0.02$) in g_0 over the six conditions ($t = 3.72$, $df = 5$). The g_0 values were generally at the same level for the pre-flight and recovery sessions, with none of the differences being significant.

5.4.2 Fits to Average Responses

Optimal model fits were also calculated for the average pre-flight, return, and recovery SPV responses for each subject. Subjects M and P did not yield analyzable SPV responses during the return sessions, for reasons which have been previously described. Subject M did not perform any dumping runs.

The parameters for the optimal model fits are shown in Tables 5.32 through 5.34, along with the mean square error (MSE) between the model response and the actual SPV data, and the number of iterations (#iter) required for the optimization routine to

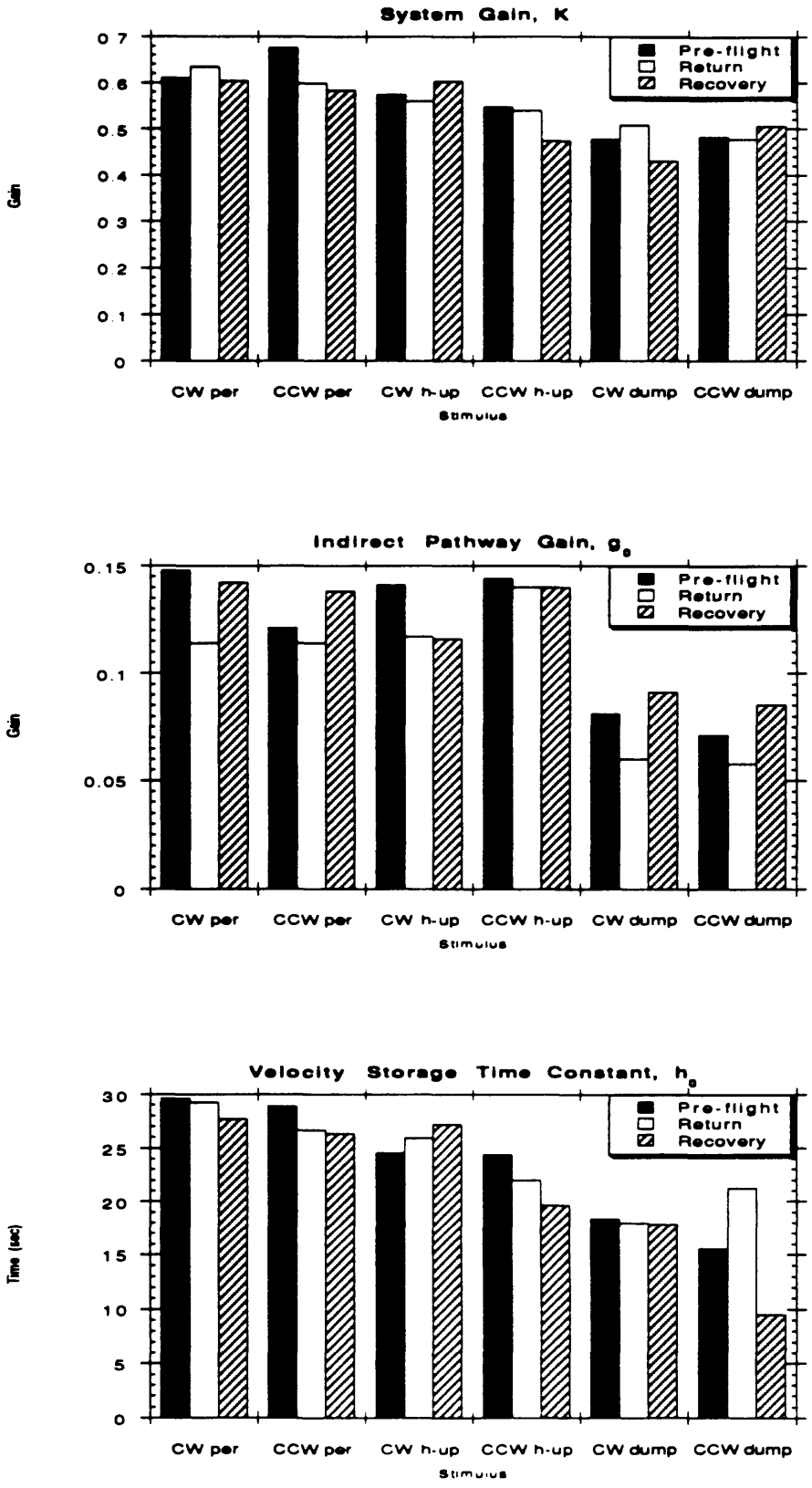


Figure 5.13. Average free parameters from model fits to individual runs for Subject N.

converge to the solution. *MSE* and *#iter* are included so that the quality of the fits may be compared, particularly the differences between subjects.

Even with exceptionally good data such as that of Subject N, the parametric fits to the average SPV data were not the same as the average of the parametric fits to the individual runs. The trends found in Section 5.4.1 were mostly present in the average SPV data to some extent; however, the details were often different. The potential complications were indicated by the fact that the gain K was larger for CW head-up post-rotatory data than for per-rotatory data during the pre-flight and recovery sessions. When the individual runs were fit and the parameters averaged, the opposite result (per-rotatory K larger than post-rotatory K) was consistent across all sessions and both directions.

Another problem with parameterizing average data was evident in the dumping runs for subject M, where g_0 reached the upper bound of 0.45 in both directions. When individual runs were fit for subject N, such runs were discarded and the average parameters were based upon the remaining data. Such a luxury was not possible for subject M. Seven other cases where g_0 reached unusually high values ($g_0 > 0.2$) have been underlined. The proliferance of these cases made it difficult to draw any conclusions about the SPV responses of the other subjects. The combination of these missing data, and the variability from subject to subject, made it impossible to summarize the results pictorially as was done in Section 5.4.1.

The directional asymmetry in subject T was one result which was apparent in the optimal parameters. The pre-flight gains K and g_0 were noticeably different between the CW and CCW directions, with a 24% asymmetry in K. However, these parameters were virtually identical for the return data. There was a 28% directional asymmetry in

K for the recovery data, and the difference in g_0 also returned. It was impossible to reach statistical conclusions about these differences since there was no variance information, but it correlated well with the results of the "sum of t-square" analysis.

Subj	Dir	Time	K	g_0	$1/h_0$	MSE	#iter
M	CW	per	0.540	0.104	23.52	6.96	84
		h-up post	0.405	0.150	21.76	12.83	95
		dump post	0.157	0.450	6.74	8.07	115
	CCW	per	0.563	0.127	25.77	11.96	94
		h-up post	0.347	0.210	14.08	9.59	141
		dump post	0.116	0.450	7.11	9.65	149
N	CW	per	0.587	0.150	29.35	1.80	60
		h-up post	0.608	0.130	23.71	1.83	85
		dump post	0.457	0.085	17.69	2.19	93
	CCW	per	0.674	0.120	28.80	1.75	57
		h-up post	0.549	0.145	23.68	1.44	77
		dump post	0.520	0.063	17.51	2.39	98
P	CW	per	0.626	0.077	31.09	6.58	98
		h-up post	0.511	0.097	23.15	5.80	90
		dump post	0.310	0.045	15.94	8.34	182
	CCW	per	0.541	0.090	27.53	7.00	78
		h-up post	0.266	0.292	13.13	32.60	142
		dump post	0.180	0.151	10.68	11.09	122
T	CW	per	0.451	0.132	29.46	6.17	89
		h-up post	0.509	0.145	40.43	8.59	67
		dump post	0.315	0.115	21.21	3.79	98
	CCW	per	0.576	0.119	45.79	3.54	73
		h-up post	0.183	0.343	24.61	7.74	210
		dump post	0.237	0.148	15.85	4.06	99

Table 5.32. Optimal three-parameter model fits to pre-flight SPV data. Explanation of entries in text.

Subj	Dir	Time	K	g0	1/h0	MSE	#iter
N	CW	per	0.615	0.114	29.10	2.27	81
		h-up post	0.541	0.116	25.17	10.49	85
		dump post	0.502	0.059	18.16	3.13	103
	CCW	per	0.611	0.108	26.51	3.88	67
		h-up post	0.584	0.121	22.96	5.77	69
		dump post	0.487	0.046	15.42	3.75	98
T	CW	per	0.573	0.111	33.66	10.19	73
		h-up post	0.569	0.102	60.44	32.51	77
		dump post	0.183	0.264	21.70	13.72	174
	CCW	per	0.562	0.120	46.92	8.72	73
		h-up post	0.408	0.178	29.48	31.76	66
		dump post	0.151	0.449	12.91	13.81	129

Table 5.33. Optimal three-parameter model fits to return SPV data. Explanation of entries in text.

Subj	Dir	Time	K	g0	1/h0	MSE	#iter
M	CW	per	0.540	0.108	20.28	12.19	88
		h-up post	0.444	0.108	19.17	11.99	105
		dump post	—	—	—	—	—
	CCW	per	0.362	0.173	20.06	16.45	134
		h-up post	0.323	0.143	14.55	4.64	109
		dump post	—	—	—	—	—
N	CW	per	0.598	0.138	27.44	3.76	71
		h-up post	0.662	0.096	28.48	10.35	63
		dump post	0.436	0.084	18.12	2.01	93
	CCW	per	0.576	0.120	27.66	6.32	74
		h-up post	0.454	0.175	18.17	5.02	88
		dump post	0.500	0.083	9.56	3.12	96
P	CW	per	0.436	0.084	24.96	11.82	107
		h-up post	0.407	0.079	18.22	10.68	123
		dump post	0.427	0.010	120.89	12.54	244
	CCW	per	0.561	0.064	29.45	5.76	127
		h-up post	0.404	0.095	17.64	11.99	111
		dump post	0.262	0.097	5.65	15.70	125
T	CW	per	0.323	0.167	25.93	8.29	97
		h-up post	0.100	0.434	43.67	17.93	169
		dump post	0.178	0.250	16.16	5.08	156
	CCW	per	0.427	0.128	42.95	18.40	87
		h-up post	0.223	0.219	27.34	21.19	148
		dump post	0.352	0.091	16.56	5.18	86

Table 5.34. Optimal three-parameter model fits to recovery SPV data. Explanation of entries in text.

6. Conclusions

The horizontal angular vestibulo-ocular reflex was investigated in the payload and mission specialists on the Spacelab Space Life Sciences 1 mission, during ground tests before and immediately after the shuttle mission. The physical stimulus to the vestibular system consisted of an exponential ramp (time constant of 0.17 sec) to a constant horizontal angular velocity of 120 °/s for a period of one minute while the subjects were seated with the head up-right, followed by an identical ramp down to zero velocity. In one-half of the runs (head-up), the head was held upright after the chair stop; in the other half (dumping), the head was pitched forward 90° immediately after the stop. Eye position was monitored by EOG for the one minute spin and for one minute subsequent to the stop. The slow phase velocity (SPV) was calculated from the eye position by a series of programs which were developed in Think C and MatLab. The SPV is considered to be indicative of the central nervous system's estimate of the body's rate of rotation. Subjective sensations were recorded by having the astronauts press a button to mark the time at which the sensation of rotation became ambiguous in direction, or disappeared entirely; theoretically, these times should correlate with the zero crossings of the SPV.

The data analysis system which was developed was almost fully automated. An interactive operator was only routinely required for determination of the calibration factors. Unusual circumstances (for instance, a button push overlapping the chair stop) would occasionally cause the automated program to fail, so that operator intervention would be needed. The C implementation of OS filters permitted the calculation of the SPV faster than real time, with no assumptions being made about the stimulus or the response. The automated outlier detection algorithm and the model fitting procedure naturally made assumptions about the form of the stimulus and the response; however,

the remainder of the statistical analysis did not. The overall system produced scientifically and statistically reliable results which appeared as good as, or better than, the results obtained through other methods. In particular, the outlier detection and removal algorithm for treating dropouts, the simulation of the sum-of-t-squares, and the five-parameter model fitting all treated the data in a more realistic manner than had similar previous experiments.

Several significant findings arose from the use of a robust and uniform data analysis procedure. Some of the results agreed with previous experiments, while others did not, and still more were totally unexpected.

In all four subjects, the dumping head movement produced a highly significant reduction in the post-rotatory SPV, as compared to the post-rotatory SPV from head-up runs. This difference was also exhibited by a significantly shorter duration of subjective sensations. The model fits to Subject N's data demonstrated a consistent decrease in the system gain K, the indirect pathway gain g_0, and the velocity storage time constant $1/h_0$. All three parameters were significantly shortened in both directions for the pre-flight data. The head movement produced a large sensory conflict which appeared to be resolved by decreasing all three parameters simultaneously, rather than any individual parameter. Figure 6.1 shows the magnitudes of the post-rotatory button push times and the three free model parameters, for each of head-up and dumping runs. The data was averaged across pre-flight CW and CCW runs, and plotted relative to the head-up level, which demonstrates that the largest relative decrease was in g_0. The actual average values are indicated atop the vertical bars.

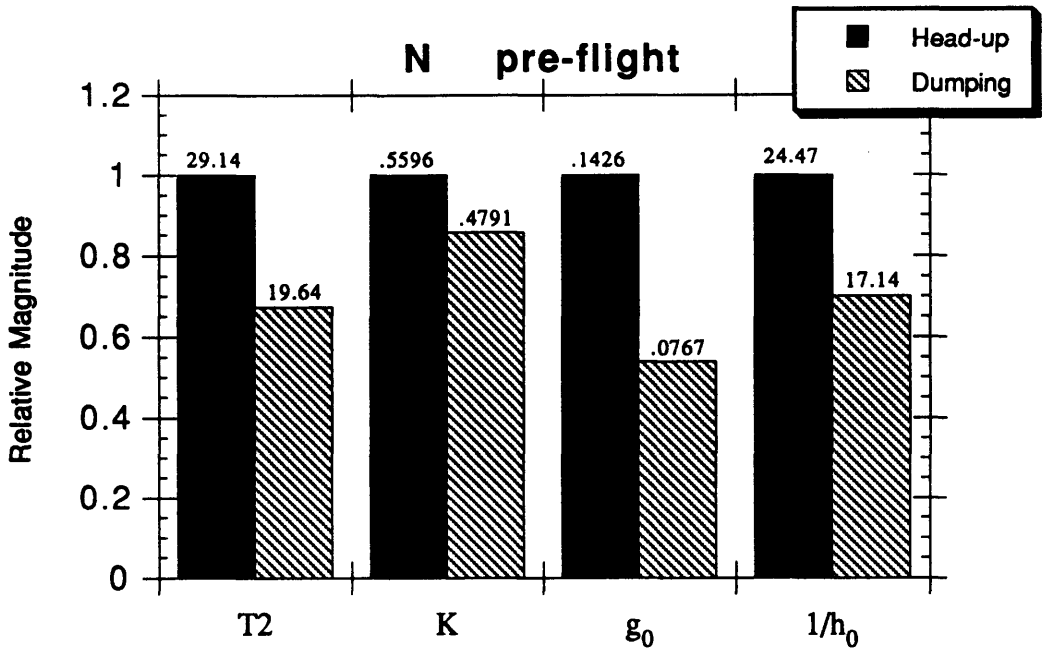


Figure 6.1. Differences between post-rotatory head-up and dumping responses for Subject N, pre-flight. Vertical bars are normalized to the head-up level. Actual average values are recorded above the bars. Data are averaged across both directions.

In three of the four subjects, the head-up post-rotatory SPV was significantly higher than the per-rotatory SPV as calculated by a sum of t-squares test. However, the post-rotatory sensation durations were generally shorter than, or equivalent to, the per-rotatory times. Indeed, the model fits (one subject) showed that K and $1/h_0$ were both significantly lower for the post-rotatory data, while g_0 was unaffected. Therefore, the increase in the magnitude of the SPV must be attributed to the reversals caused by the neural adaptation term; this produces a non-zero SPV at the beginning of the post-rotatory data which acts as an offset. It had originally been hoped to combine the per- and post-rotatory responses to improve the statistical analysis. However, this shows that comparisons between per-rotatory and post-rotatory SPV data must be made with great care.

The post-flight SPV was generally lower or unchanged from the pre-flight baseline levels. During the return sessions (within two days of return to Earth), the SPV in one subject (N) was significantly reduced from the pre-flight baseline for per-rotatory data, with visible, but not significant, decreases in the post-rotatory data. During the recovery sessions, the SPV profiles for that subject were still reduced from pre-flight levels, but the differences were not as great as for the return sessions. The button push times also followed this pattern of reduction immediately post-flight, followed by partial recovery. The model fits to the individual runs showed no significant or consistent changes in K or $1/h_0$; however, g_0 was significantly reduced from pre-flight levels during the return sessions, and was equivalent to pre-flight levels during the recovery sessions. Figure 6.2 shows average values for the button push time and three free model parameters for Subject N for each of pre-flight, return, and recovery sessions. These values were obtained by averaging across all six stimulus conditions (all combinations of CW and CCW directions, per- and post-rotatory, head-up and dumping). The magnitudes are plotted relative to the pre-flight baseline level, with the actual average values indicated atop the vertical bars. The subjective sensations and the indirect pathway gain followed the pattern of increases and decreases in the SPV, and these changes were significant. The changes in K and h_0 were small and not statistically significant, and did not follow the same pattern.

The behaviour of the model parameters suggests that only the indirect pathway gain is modified by the CNS to compensate for the difference in gravito-inertial force level, and is returned to its normal pre-flight 1 G level as the body readapts to the 1 G environment. The change in T is a result of this modification process. It is possible, however, that K and $1/h_0$ are also modified and that their time course of readaptation is faster than that of g_0. Perhaps, if N had been tested on the day of the landing, this could have been determined. It should be emphasized that this is contrary to the

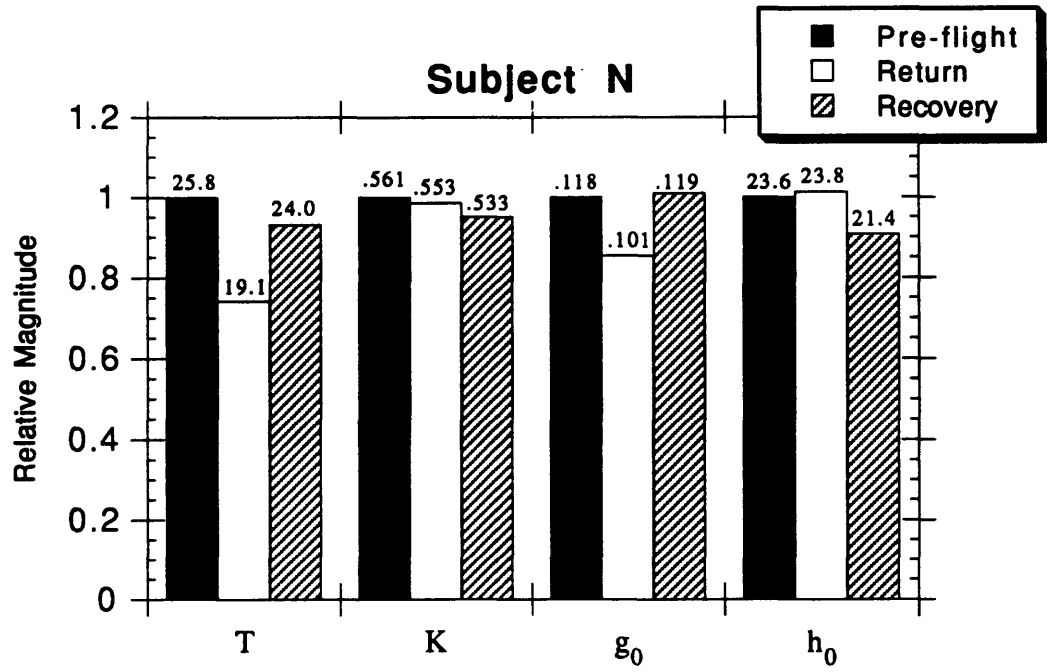


Figure 6.2. Differences amongst pre-flight, return, and recovery responses for N. Vertical bars are normalized to the pre-flight level. Actual average values are recorded above the bars. Data are averaged across both directions, per- and post-rotatory, and head-up and dumping.

suggestion of Raphan that the change would be manifested in the leak rate, h_0. While decreases in g_0 and $1/h_0$ produce qualitatively similar decreases in the SPV, model fits to this data set selected g_0 as the source of the quantitative change.

Two other subjects (M and P) did not yield any analyzable SPV data for the return sessions. The reasons for this were likely a combination of pain and medication for M, nausea for P, and overall drowsiness for both which resulted in long periods of no VOR response. Although the SPV was generally, and occasionally significantly, reduced from the pre-flight levels during the recovery sessions, the problems with dropouts persisted to a lesser extent and may have been responsible for this reduction. The button push times for M were slightly reduced from pre-flight levels on the return

days, but not significantly lower on the recovery days; the button push times for P were unchanged from pre-flight to post-flight. No pattern was apparent in the model fit parameters for these subjects. For M and P, there is neither support for, nor contradiction of, the results for N.

The fourth subject (T) produced responses which were quite different from the others', due in part to directional asymmetries in the data. The horizontal calibrations were asymmetric throughout all sessions, both pre-flight and post-flight; the cause of this asymmetry was thought to be oculomotor in nature, due to esophoria. The SPV also showed a directional asymmetry pre-flight, with left-beating nystagmus producing a higher SPV than right-beating. Figure 6.3 portrays the system gain K as determined by optimal model fits to the average per-rotatory SPV for Subject T. K is shown for both CW and CCW directions, for each of the pre-flight, return, and recovery sessions. The model fits showed an asymmetry of 24% in the system gain K. However, during the return sessions, the right-beating nystagmus was significantly increased from pre-flight levels, while the left-beating SPV was not significantly different from pre-flight. This resulted in an apparent disappearance of the asymmetry. In particular, K was virtually identical for both directions. During the recovery sessions, the directional asymmetry had reappeared in both the SPV profile (sum of t-squares test) and the model parameter K (28% asymmetry). Although K was reduced from pre-flight levels in both directions, this was likely due to fatigue. There was no corresponding significant change in the durations of subjective sensations.

It is evident that this asymmetry is not related to the calibration asymmetry. Therefore, it appears that the asymmetry in K is controlled by some mechanism which is switched off when the body is exposed to 1 G after weightlessness, but reappears after the body readapts to the 1 G environment.

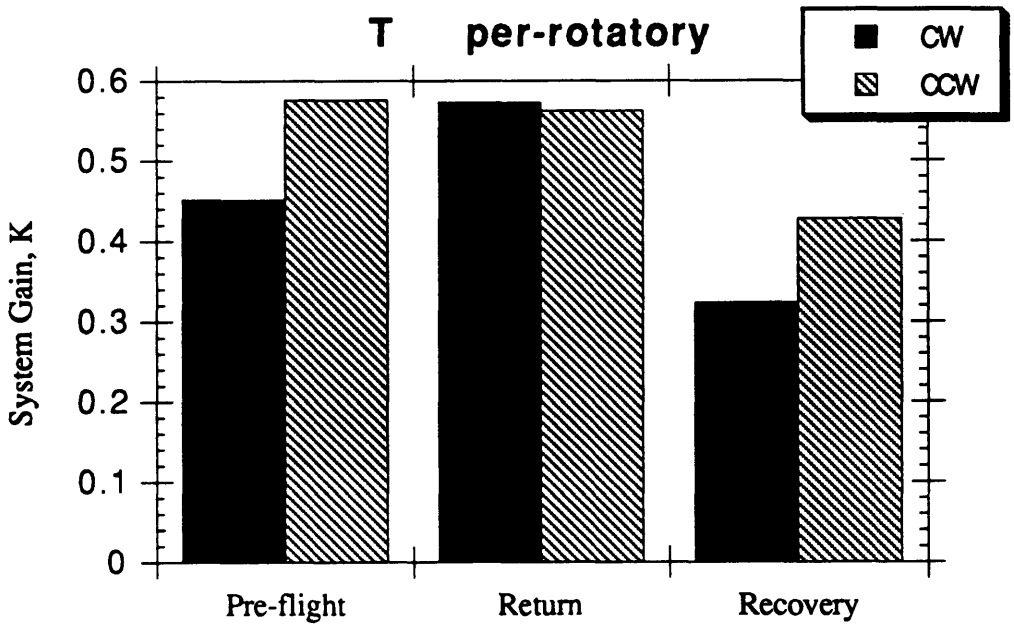


Figure 6.3. Change in directional asymmetry for Subject T across sessions. The system gain, K, is plotted for CW and CCW runs, for each of the pre-flight, return, and recovery sessions.

Overall, the results agree in principle with those obtained by our laboratory on previous shuttle missions. The differences between pre- and post-flight SPV profiles were not as large for SLS-1 data as for SL-1 and D-1; however, this may have been due to the use of a more appropriate (and more stringent) statistical test, and a uniform method of dealing with dropouts. The nature of the reduction in SPV upon exposure to a different GIF has been traced to the indirect pathway gain. Although this is based upon only one subject, the model fits were quite good and thus lend general support to the Raphan-Cohen model when modified to include neural adaptation. Post-flight sensory conflict may be physically quite different from that caused by the dumping head movements, since the dumping conflict only persists for about 20 seconds; indeed, the changes in the theoretical model parameters are also quite different.

6.1 Recommendations for Future Work

These results pose many more questions than they answer. Is g_0 the only parameter that changes upon return to Earth? What happened to the asymmetry in the SPV for T? What mechanism controls it? Did the dumping head movement produce a build-up of torsional nystagmus? What happened to the asymmetry in the per-rotatory sensation durations for M? Did the calibration asymmetry in T affect the SPV data? If so, how?

These require additional experiments. One obvious choice is to test T again, recording the horizontal movements of each eye individually. This will determine whether the eyes move disconjugately in the dark with no fixation points. If there is disconjugate motion, this will also better show whether there is a change in the disconjugacy after exposure to different GIFs.

The dumping effects could be better determined by using search coils or video recording techniques so that all three axes of eye movement could be simultaneously recorded. The pitching head movement which is currently used should produce a faster decay of the horizontal SPV and a transient vertical nystagmus during the head motion. A central research question is whether any nystagmus occurs about the original axis of rotation, in this case the earth-vertical (see Section 2.4). With the pitch movement, the potential nystagmus would be torsional, which can not be measured by EOG. Alternatively, the pitching head movement could be replaced by a roll movement so that any buildup of nystagmus would be vertical, allowing monitoring by EOG. The accurate calculation of the magnitude of any such nystagmus would naturally require more investigation into the calibration of the non-linearity in the vertical EOG potential, as well as into the non-linear vertical SPV responses which have been observed by various researchers.

More data should also be obtained regarding the changes in the horizontal EOG potential over time under different lighting conditions. The study by Gonshor and Malcolm (1971) was on a small number of subjects. If dark adaptation by exposure to red light for a certain period of time is indeed sufficient for the EOG gain to reach a steady state, the determination of the length of time is critical.

A very important question is as to the changes, if any, which occurred in the SPV response in orbit. This data should be processed, and compared to the pre-flight data. It would be beneficial to obtain repetitions on the in-flight data. This will have to wait until IML-1 in 1992 and SLS-2 in 1993.

A method should be found to improve the level of alertness of the crew, particularly during the post-flight sessions when the high demands on the crew and the small amount of sleep in orbit result in very tired subjects. Experiments should be performed in which a tired subject is tested with and without an alerting technique to determine the efficacy of the technique. One technique which seems to have been overlooked is the playing of music which has been selected by the subject.

A number of modifications to the data analysis software are also suggested. The AATM algorithm, as it stands, only processes a single channel of eye position data at a time. The multi-dimensional approach used by Merfeld (1990) is superior in performance to the equivalent one-dimensional algorithm. It would be interesting to develop a multi-dimensional version of the AATM algorithm, operating on the magnitude of the eye velocity vector.

Since AATM does not perform well with data containing sudden jumps in SPV, a hybrid algorithm may be superior. Within a couple of seconds of a sharp increase in the stimulus, such as a step in angular velocity, a different algorithm (perhaps Merfeld's acceleration-based algorithm) could be used which performs well under such conditions. AATM could be run on the remainder of the data set.

Alternatively, AATM could be implemented as a two-pass algorithm. After running AATM once, the differences between the resultant SPV and the original eye velocity could be examined. Any differences exceeding some threshold (say 200 °/s) would be classified as a fast phase and discarded. AATM would then be re-run on the remaining data. An obvious problem with this approach is that it is a return to the concept of user-specified thresholds.

As yet another alternative, AATM could written in a more "implicit" form. This would involve replacing velocity values by the SPV values as they are calculated. Therefore, AATM's estimate of the SPV at any given point in time would be based upon 0.5 sec of raw velocity values and 0.5 sec of neighbouring SPV values. The information on which the SPV estimate is based should then be even more reliable, although the underlying theoretical statistics would likely become more complicated. The advantage of such an approach, if it works, is to run the implicit algorithm on the data forwards and backwards, and see if it improves the performance on those areas with sudden jumps in SPV.

The entire data analysis system is very well-automated, but a user interface is required to tie all the pieces together. The algorithms are also independent of the stimulus, except for the outlier detection and model fitting. The model fitting should be changed to incorporate the actual recorded stimulus velocity as the input to the model, although

this can become more complicated with caloric tests. The outlier detection algorithm essentially does a simplistic model fit; it too could be changed to accept the actual stimulus velocity as an input. The goal of this would be to develop a data analysis package that does not require any *a priori* information as to the type of stimulus used, or as to the type of response expected — a fully automated, threshold-independent, unbiased, model-fitting system. Such a system could hopefully be used equally well in research institutions and in diagnostic clinics.

References

Allum, J.H.J., J.R. Tole, and A.D. Weiss. MITNYS-II — A Digital Program for On-Line Analysis of Nystagmus. *IEEE Transactions on Biomedical Engineering*, 22 (3):196-202, 1975.

Anzaldi, E. and E. Mira. An Interactive Program for the Analysis of ENG Tracings. *Acta Otolaryngologica*, 80:120-127, 1975.

Baker, R. and A. Berthoz. Control of Gaze by Brain Stem Neurons. Developments in Neuroscience, Volume 1. Elsevier/North-Holland, New York, NY, 1977.

Baloh, R.W., V. Honrubia, and H.R. Konrad. Ewald's Second Law Re-evaluated. *Acta Otolaryngologica*, 83:475-479, 1977.

Baloh, R.W., L. Langhofer, V. Honrubia, and R.D. Yee. On-line Analysis of Eye Movements Using a Digital Computer. *Aviation, Space, and Environmental Medicine*, 51(6):563-567, 1980.

Breiman, L. Statistics: With a View Toward Applications. Houghton Mifflin Company, Boston, MA, 1973.

Cheney, W. and D. Kincaid. Numerical Mathematics and Computing, Second Edition. Wadsworth, Inc., Belmont, CA, 1985.

Cohen, B., V. Matsuo, and T. Raphan. Quantitative Analysis of the Velocity Characteristics of Optokinetic Nystagmus and Optokinetic After-nystagmus. *Journal of Physiology*, 270:321-344, 1977.

Cohen, B., T. Uemura, and S. Takemori. Effects of Labyrinthectomy on Optokinetic Nystagmus (OKN) and Optokinetic After-nystagmus (OKAN). *Equil. Res.*, 3:88-93, 1973.

Davis, J.R., J.M. Vanderploeg, P.A. Santy, R.T. Jennings, and D.F. Stewart. Space Motion Sickness During 24 Flights of the Space Shuttle. *Aviation, Space and Environmental Medicine*, 59:1158-1162, 1988.

Dizio, P., J.R. Lackner, and J.N. Evanoff. The Influence of Gravitoinertial Force Level on Oculomotor and Perceptual responses to Sudden Stop Stimulation. *Aviation, Space, and Environmental Medicine*, 58 (9):A224-A230, 1987.

Dizio, P. and J.R. Lackner. The Effects of Gravitoinertial Force Level and Head Movements on Post-rotational Nystagmus and Illusory After-rotation. *Experimental Brain Research*, 70:485-495, 1988.

Engelken, E.J. and K.W. Stevens. A New Approach to the Analysis of Nystagmus: An Application for Order-Statistic Filters. *Aviation, Space, and Environmental Medicine*, 61 (9):859-864, 1990.

Engelken, E.J., K.W. Stevens, and J.D. Enderle. Optimization of an Adaptive Nonlinear Filter for the Analysis of Nystagmus. Rocky Mountain Bioengineering Symposium, 1991.

Gillingham, K.K. and J.W. Wolfe. Spatial Orientation in Flight. USAF Technical Report USAFSAM-TR-85-31, 1986.

Goldberg, J.M. and C. Fernández. The Vestibular System. In: Handbook of Physiology, Chapter 21, pp. 977-1022.

Gonshor, A. and R. Malcolm. Effect of Changes in Illumination Level on Electro-oculography (EOG). *Aerospace Medicine*, 42 (2):138-140, 1971.

Grace, A. Optimization Toolbox User's Guide. The MathWorks, Inc., Natick, MA, 1990.

Guedry, F.E., Jr. and A.J. Benson. Coriolis Cross-coupling Effects: Disorientating and Nauseogenic or Not? *Aviation, Space, and Environmental Medicine*, 49 (1):29-35, 1978.

Heinonen, P. and Y. Nuevo. FIR-meadian Hybrid Filters. *IEEE Transactions on Acoustics and Speech Signal Processing*, 35:832-838, 1987.

Henn, V., B. Cohen, and L.R. Young. Visual-Vestibular Interaction in Motion Perception and the Generation of Nystagmus. Neurosciences Research Program Bulletin. MIT Press, Cambridge, MA, 1980.

Hogg, R.V. Adaptive Robust Procedures: A Partial Review and Some Suggestions for Future Applications and Theory. *Journal of the American Statistical Association*, 69:909-923, 1974.

Huber, P.J. Robust Statistics. Wiley Press, NY, 1981.

Johnson, K.D. and J. Gidney, Jr. The Design and Construction of a Revolving Chair: Measuring the Effects of Weightlessness on the Vestibular System. 16.622 report, Massachusetts Institute of Technology, Cambridge, MA, 1983.

Kulbaski, M.J. Effects of Weightlessness on the Vestibulo-Ocular Reflex in the Crew of Spacelab 1. BSME Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1986.

Lackner, J.R. and A. Graybiel. Elicitation of Motion Sickness by Head Movements in the Microgravity Phase of Parabolic Flight Maneuvers. *Aviation, Space, and Environmental Medicine*, 55:513-520, 1984.

Laurence Urdang Associates, Ltd. The Bantam Medical Dictionary. Bantam Books, New York, NY, 1982.

Massoumnia, M. Detection of Fast Phase of Nystagmus Using Digital Filtering. SM Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1983.

Merfeld, D.M. Spatial Orientation in the Squirrel Monkey: An Experimental and Theoretical Investigation. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1990.

Michaels, D.L. Microprocessor Analysis of Eye Movements. SM Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1977.

Oman, C.M. and M.J. Kulbaski. Spaceflight Affects the 1-g Postrotatory Vestibulo-ocular Reflex. *Adv. Oto-Rhino-Laryng.* 42:5-8, 1988.

Oman, C.M., B.K. Lichtenberg, K.E. Money, and R.K. McCoy. MIT/Canadian Vestibular Experiments on Spacelab-1: 4. Space Motion Sickness: Symptoms, Stimuli, and Predictability. *Experimental Brain Research*, 64 (2):316-334, 1986.

Oman, C.M. and H. Weigl. Postflight Vestibulo-ocular Reflex Changes in Space Shuttle/Spacelab D-1 Crew. Aerospace Medical Association abstract, 1989.

Pouliot, C.F. Summary of the Sigma T-squared Simulation. Massachusetts Institute of Technology Man-Vehicle Lab internal report, 1991.

Raphan, T., B. Cohen, and V. Matsuo. A Velocity Storage Mechanism Responsible for Optokinetic Nystagmus (OKN), optokinetic After-nystagmus (OKAN) and Vestibular Nystagmus. *In: Control of Gaze by Brain Stem Neurons*, 1977.

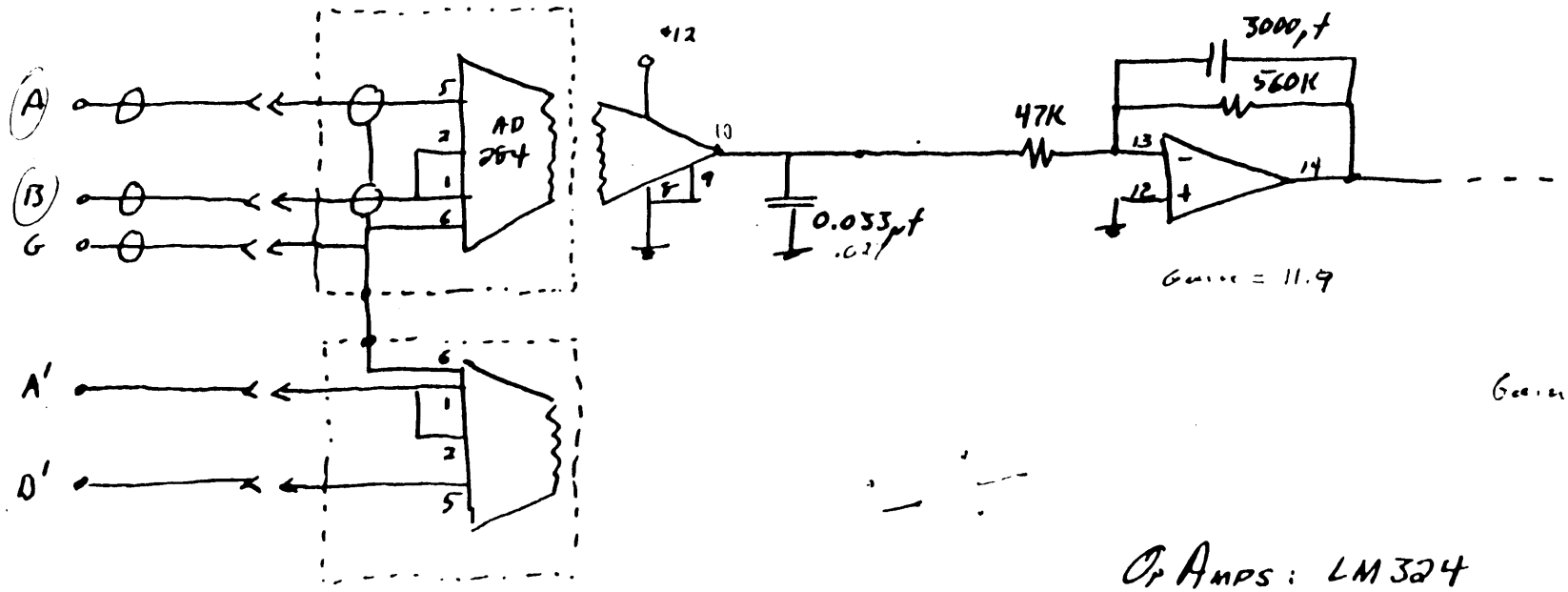
Rosner, B. Fundamentals of Biostatistics, Second Edition. PWS Publishers, Boston, MA, 1986.

Wilson, V.J. and G. Melvill Jones. Mammalian Vestibular Physiology. Plenum Press, New York, NY, 1979.

Young, L.R. and C.M. Oman. Model for Vestibular Adaptation to Horizontal Rotation. *Aerospace Medicine*, 40 (10):1076-1080, 1969.

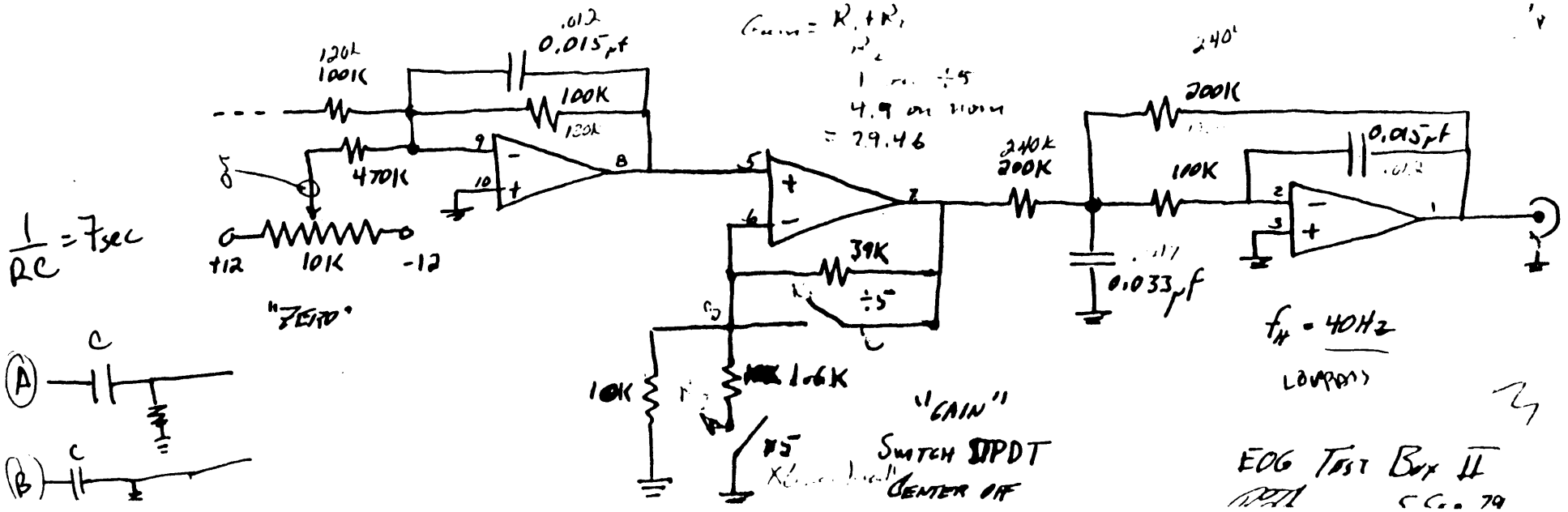
Appendix A

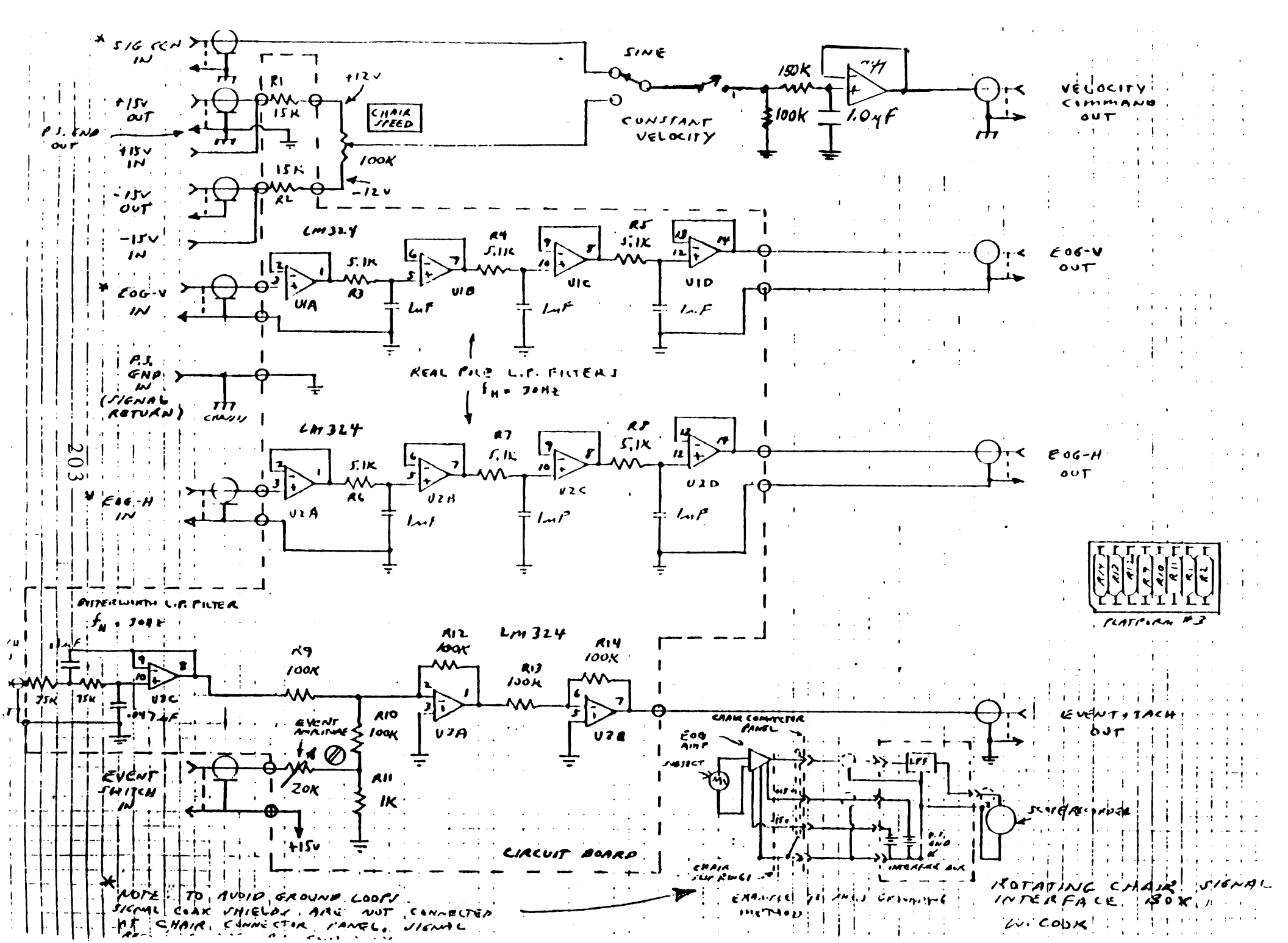
Schematic Diagrams



Gain = 119

OP AMPS: LM324
2 IDENTICAL CIRCUITS
(LABELED H & V)

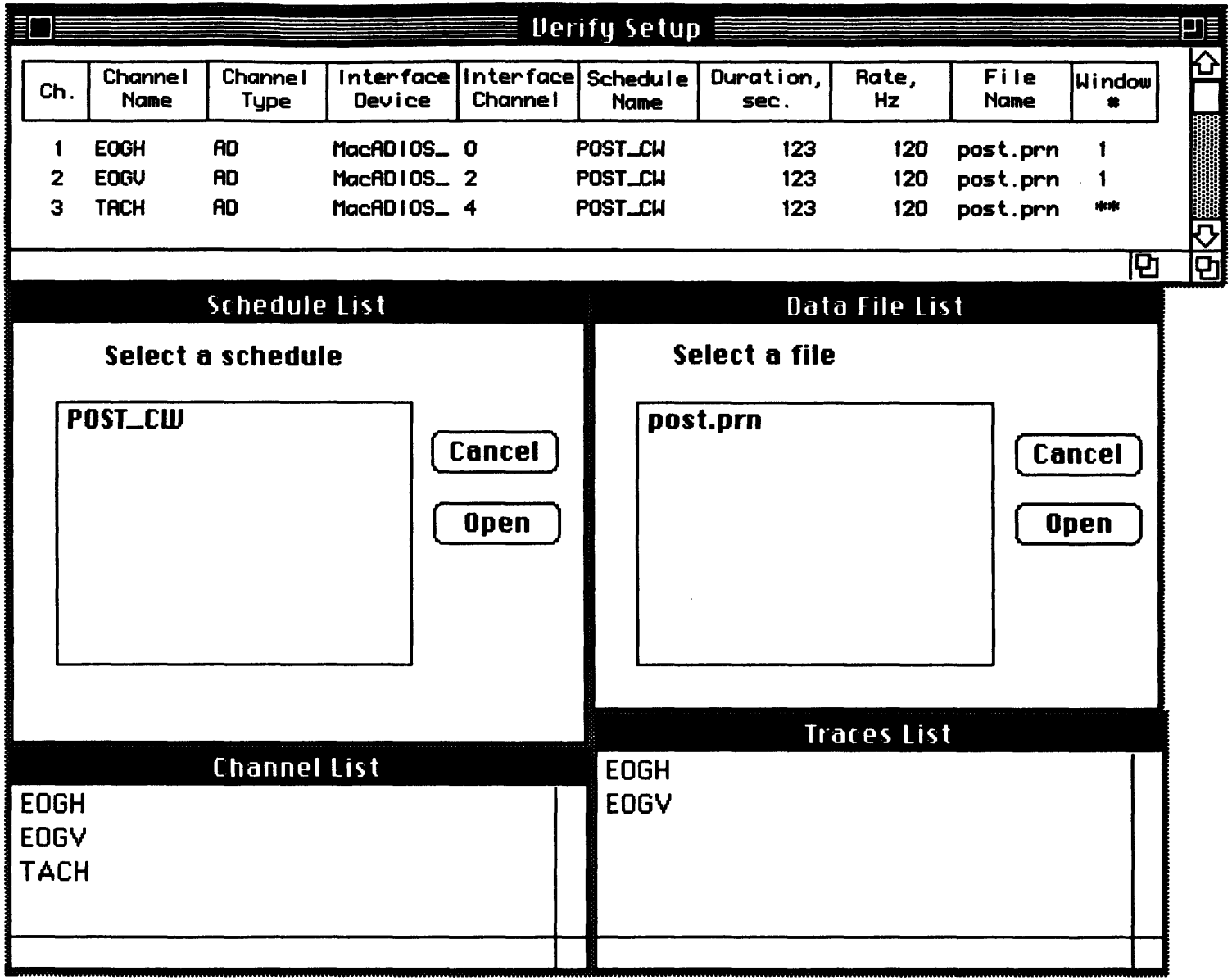




NOTE: TO AVOID GROUND LOOPS SIGNAL COAX SHIELDS ARE NOT CONNECTED AT CHAIR CONNECTOR PANEL. SIGNAL

Appendix B

LabTech Notebook Setup


Verify Setup
Ch.
Channel Name
Channel Type
Interface Device
Interface Channel
Schedule Name
Duration, sec.
Rate, Hz
File Name
Window #
1 EOGH AD MacADIOS_ 0 POST_CW 123 120 post.prn 1
2 EOGV AD MacADIOS_ 2 POST_CW 123 120 post.prn 1
3 TACH AD MacADIOS_ 4 POST_CW 123 120 post.prn **
Schedule List
Select a schedule
POST_CW
Cancel
Open
Data File List
Select a file
post.prn
Cancel
Open
Channel List
EOGH
EOGV
TACH
Traces List
EOGH
EOGV

Appendix C

Data Format Conversion Software

BDCF_convert.c

edit_header.c

Notes on LabTech to MatLab Data Conversion
D. Balkwill 12/22/91

The data for the SLS-1 rotating chair experiment is collected via LabTech Notebook, while the detailed analysis is done in MatLab. Since the data file formats for these two applications are not equivalent, a conversion must take place. Two types of programs have been written in Think C to perform this conversion. All programs aree linked with the *ANSI* and *unix* libraries.

The first program to run is *BDCF_convert*, which converts the data from LabTech Notebook format (a matrix of binary real numbers) to MatLab format. During these sessions, three channels (EOG H, EOG V, and TACH) were saved in a single file as binary real numbers. Binary format was chosen instead of ASCII to conserve disk space and the time required to save the data. *BDCF_convert* reads a binary file whose name is specified by the user, in blocks of 32K bytes. The three channels of data are moved into three separate buffers and the data values are scaled from voltage readings (real numbers between -10 V and +0.995 V) to two-byte integer A/D units (between -2048 and +2047), to conserve disk space. Each channel is saved in its own file, whose name is obtained by appending the appropriate extension (*.eogh*, *.eogv* or *.tach*) to the user-specified file name. Each file consists of the binary data values, preceded by a header which includes the data type, numbers of rows and columns, imaginary data flag, and the variable name and length. The data type for the rotating chair data is set at 1030 to reflect two-byte integer data saved on a Macintosh. Each channel is a time series of samples which is saved as a column vector, setting the number of columns to 1 so that the number of rows is the same as the number of samples. The imaginary flag is set to 0 to indicate real data. The variable name is either *eogh*, *eogv*, or *tach*. The program loops for as many files as the user may wish to edit.

It was discovered that, in a few cases, the number of samples saved by *BDCF_convert* was incorrect. After extensive debugging, it was determined that the reason for this error was not in the conversion program; the reason remains a mystery still. As such, a second program *edit_header* was written to display the header information in a user-specified file and allow the user to change these values if desired. However, the variable name cannot be changed since it is too complicated to actually increase the number of bytes in the header; the main purpose of the program was to correct the errors made in the saving of the header. The user merely enters a number between 1 and 5 corresponding to the piece of data which is to be changed, and then enters the new value; when all parameters have been properly set, entering 0 saves the new header. The program loops for as many files as the user may wish to edit.

LabTech Notebook frequently crashed, giving an error message of *"Error 300, Status 1"*, indicating that the data acquisition software could not keep up with the amount of data which was required to be sampled. In an attempt to stretch the capabilities of the software, the setup for the data acquisition was frequently redone. This allowed the data to be collected, but it resulted in three different data formats. For BDC5, the order of the two EOG channels was reversed. For the post-flight sessions (BDC6 through BDC10), the channels were saved in the correct order, but the data were saved directly as two-byte integer A/D units. Two programs, *BDC5_convert* and *post_convert* were written to convert this data; they were minor modifications of *BDCF_convert*. The order of the channels and the type of data are changed simply by modifying the *row* struct definition at the beginning of the program. The scaling step (multiplication by *A2D_SCALE* and addition of 0.5) was removed for post-flight data since the data was already in two-byte integer format.

```
/* BDCF_convert.c */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unix.h>

#define FALSE 0
#define BLOCK_SIZE  (32767/sizeof(row))
#define A2D_SCALE  204.8
#define MATLAB_TYPE 1030
#define FILE_NAME_LENGTH 81
#define EOGH_NAME "eogh"
#define EOGV_NAME "eogv"
#define TACH_NAME "tach"

typedef struct {
        float eogh;
        float eogv;
        float tach;
        } row;


char in_filename[FILE_NAME_LENGTH];
int in_handle;
row *in_buffer;
int num_in_bytes;

char eogh_filename[FILE_NAME_LENGTH];
int eogh_handle;
FILE *eogh_fptr;
short *eogh_buffer;

char eogv_filename[FILE_NAME_LENGTH];
int eogv_handle;
FILE *eogv_fptr;
short *eogv_buffer;

char tach_filename[FILE_NAME_LENGTH];
int tach_handle;
FILE *tach_fptr;
short *tach_buffer;

int num_out_bytes;

typedef struct {
        long type;
        long mrows;
        long ncols;
        long imagf;
        long namlen;
        } Fmatrix;

Fmatrix F_eogh, F_eogv, F_tach;
```

```
long mrows = 0L;
char in_line[FILE_NAME_LENGTH];

main()
{
        int ch;
        int i,j,k;


/* allocate memory for buffers */
        in_buffer = (row *)malloc(BLOCK_SIZE * sizeof(row));
        if (!in_buffer) {
                printf("out of memory on in_buffer allocation\n");
                goto done;
                }
        eogh_buffer = (short *)malloc(BLOCK_SIZE * sizeof(short));
        if (!eogh_buffer) {
                printf("out of memory on eogh_buffer allocation\n");
                goto done;
                }
        eogv_buffer = (short *)malloc(BLOCK_SIZE * sizeof(short));
        if (!eogv_buffer) {
                printf("out of memory on eogv_buffer allocation\n");
                goto done;
                }
        tach_buffer = (short *)malloc(BLOCK_SIZE * sizeof(short));
        if (!tach_buffer) {
                printf("out of memory on tach_buffer allocation\n");
                goto done;
                }

        do {

/* get input filename and open files */
                do {
                        printf("Enter input file name:");
                        gets(in_filename);
                        in_handle = open(in_filename,O_BINARY|O_RDONLY);
                        }
                while (in_handle <= 0);

                strcpy( (char *)eogh_filename, (char *)in_filename );
                strcat( (char *)eogh_filename, ".eogh" );
                eogh_fptr = fopen(eogh_filename,"wt");
                strcpy( (char *)eogv_filename, (char *)in_filename );
                strcat( (char *)eogv_filename, ".eogv" );
                eogv_fptr = fopen(eogv_filename,"wt");
                strcpy( (char *)tach_filename, (char *)in_filename );
                strcat( (char *)tach_filename, ".tach" );
                tach_fptr = fopen(tach_filename,"wt");

/* find length of input file */
                num_in_bytes = read( in_handle, (char *)(&(in_buffer[0])),
```

```
                                                BLOCK_SIZE * sizeof(row) );
        while (num_in_bytes > 0) {
                mrows += num_in_bytes;
                num_in_bytes = read( in_handle, (char *)(&(in_buffer[0])),
                                                BLOCK_SIZE * sizeof(row) );
                }
        printf("length of input file = %ld bytes.\n",mrows);
        mrows /= sizeof(row);
        printf("number of samples = %ld.\n",mrows);
        close(in_handle);

/* write out MatLab headers to files */

        F_eogh.type = MATLAB_TYPE;
        F_eogh.mrows = mrows;
        F_eogh.ncols = 1;
        F_eogh.imagf = FALSE;
        F_eogh.namlen = strlen(EOGH_NAME) + 1;
        fwrite( &F_eogh, sizeof(Fmatrix), 1, eogh_fptr );
        fwrite( EOGH_NAME, sizeof(char), (int)F_eogh.namlen, eogh_fptr );
        fclose(eogh_fptr);

        F_eogv.type = MATLAB_TYPE;
        F_eogv.mrows = mrows;
        F_eogv.ncols = 1;
        F_eogv.imagf = FALSE;
        F_eogv.namlen = strlen(EOGV_NAME) + 1;
        fwrite( &F_eogv, sizeof(Fmatrix), 1, eogv_fptr );
        fwrite( EOGV_NAME, sizeof(char), (int)F_eogv.namlen, eogv_fptr );
        fclose(eogv_fptr);

        F_tach.type = MATLAB_TYPE;
        F_tach.mrows = mrows;
        F_tach.ncols = 1;
        F_tach.imagf = FALSE;
        F_tach.namlen = strlen(TACH_NAME) + 1;
        fwrite( &F_tach, sizeof(Fmatrix), 1, tach_fptr );
        fwrite( TACH_NAME, sizeof(char), (int)F_tach.namlen, tach_fptr );
        fclose(tach_fptr);

/* reopen all files in low-level I/O for speedy disk access */
/* separate input file into three files, and rescale to (integer) digital counts */

        in_handle = open(in_filename,O_BINARY|O_RDONLY);
        eogh_handle = open(eogh_filename,O_BINARY|O_RDWR|O_APPEND);
        eogv_handle = open(eogv_filename,O_BINARY|O_RDWR|O_APPEND);
        tach_handle = open(tach_filename,O_BINARY|O_RDWR|O_APPEND);
        num_in_bytes = read( in_handle, (char *)(&(in_buffer[0])),
                                                BLOCK_SIZE * sizeof(row) );
        mrows = num_in_bytes / sizeof(row);
        while (mrows > 0) {
                num_out_bytes = mrows * sizeof(short);
                for (i = 0; i < mrows; i++) {
                        eogh_buffer[i] = (in_buffer[i].eogh * A2D_SCALE) + 0.5;
```

```
                        eogv_buffer[i] = (in_buffer[i].eogv * A2D_SCALE) + 0.5;
                        tach_buffer[i] = (in_buffer[i].tach * A2D_SCALE) + 0.5;
                        }
                    write( eogh_handle, (char *)eogh_buffer, num_out_bytes );
                    write( eogv_handle, (char *)eogv_buffer, num_out_bytes );
                    write( tach_handle, (char *)tach_buffer, num_out_bytes );
                    num_in_bytes = read( in_handle, (char *)(&(in_buffer[0])),
                                                    BLOCK_SIZE * sizeof(row) );
                    mrows = num_in_bytes / sizeof(row);
                    }
                close(in_handle);
                close(eogh_handle);
                close(eogv_handle);
                close(tach_handle);

/* repeat for as many files as desired */
                printf("Another file (y/n) ? [n]");
                gets(in_line);
                }
        while ((in_line[0] == 'y') || (in_line[0] == 'Y'));

done:;
}
```

```
/* edit_header.c */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unix.h>

#define FALSE 0
#define FILE_NAME_LENGTH 81

char in_filename[FILE_NAME_LENGTH];
int in_handle;
int num_in_bytes;
char var_name[FILE_NAME_LENGTH];

typedef struct {
        long type;
        long mrows;
        long ncols;
        long imagf;
        long namlen;
        } Fmatrix;

Fmatrix in_matrix;

long mrows = 0L;
char in_line[FILE_NAME_LENGTH];

main()
{
        int ch;
        int i,j,k;

        do {

/* get input filename and open file */
                do {
                        printf("Enter input file name:");
                        gets(in_filename);
                        in_handle = open(in_filename,O_BINARY|O_RDWR);
                        }
                while (in_handle <= 0);

                num_in_bytes = read( in_handle, (char *)(&in_matrix), sizeof(Fmatrix) );
                num_in_bytes = read( in_handle, (char *)(&(var_name[0])),
                                        in_matrix.namlen + 1 );

                printf("1. Matlab type = %ld\n",in_matrix.type);
                printf("2. number of rows = %ld\n",in_matrix.mrows);
                printf("3. number of columns = %ld\n",in_matrix.ncols);
                printf("4. imaginary flag = %ld\n",in_matrix.imagf);
                printf("5. variable name length = %ld",in_matrix.namlen);
                printf("   <variable name = %s>\n",(char *)var_name);
                printf("Change which variable (0 to end) ?");
```

```
                while ((ch = getchar()) != '0') {
                        fflush(stdin);
                        printf("New value: ");
                        gets(in_line);
                        i = atoi(in_line);
                        switch (ch) {
                                case '1':
                                        in_matrix.type = i;
                                        break;
                                case '2':
                                        in_matrix.mrows = i;
                                        break;
                                case '3':
                                        in_matrix.ncols = i;
                                        break;
                                case '4':
                                        in_matrix.imagf = i;
                                        break;
                                case '5':
                                        in_matrix.namlen = i;
                                        break;
                                }
                        printf("1. Matlab type = %ld\n",in_matrix.type);
                        printf("2. number of rows = %ld\n",in_matrix.mrows);
                        printf("3. number of columns = %ld\n",in_matrix.ncols);
                        printf("4. imaginary flag = %ld\n",in_matrix.imagf);
                        printf("5. variable name length = %ld",in_matrix.namlen);
                        printf("   <variable name = %s>\n",(char *)var_name);
/*
                        printf("6. variable name = %s\n",(char *)in_line);
*/
                        printf("Change which variable (0 to end) ?");
                        }
/* write out MatLab headers to files */

                lseek( in_handle, 0L, SEEK_SET );
                write( in_handle, (char *)(&in_matrix), sizeof(Fmatrix) );
                printf("File indicator = %ld\n",lseek( in_handle, 0L, SEEK_END ));
                close(in_handle);

/* repeat for as many files as desired */
                fflush(stdin);
                printf("Another file (y/n) ? [n]");
                gets(in_line);
                }
        while ((in_line[0] == 'y') || (in_line[0] == 'Y'));

done:;
}
```

Appendix D

NysA Source Code

```
%nysa_tree
%NysA V1.4 Tree Structure Directory
%D.Balkwill 8/8/91

%init --> get_path

%calibrate --> get_path
%              file_specs
%              file_name
%              clear_specs
%              three_point --> pick_regions --> InList
%                                                DeleteRow

%drift_calibrate --> get_path
%                    file_specs
%                    file_name
%                    clear_specs
%                    drift_three_point --> lin_fit
%                    pick_regions --> InList
%                                     DeleteRow

%prep_data --> get_path
%                scale_data --> file_specs
%                               file_name
%                               clear_specs
%                               condition --> OS_lin2 --> PFMH1

%process --> get_path
%              vel_filt --> filtzero
%              acc_filt --> filtzero
%              heart
%              remove --> fill

%backup_data --> get_path
%                    file_specs
%                    file_name
%                     clear_specs

%edit_spv --> get_path
%              file_specs
%              file_name
%              clear_specs
%              filtzero
%              edit_alg --> diff_list
%                           InList
%                           DeleteRow

%batch --> get_path
%            file_specs
%            file_name
%            clear_specs
%            scale_data
%            process
%            backup_data
```

The scripts for NysA are included in this Appendix in alphabetical order. The page numbers are given below. The overall package was written by D. Balkwill (me). The original fast phase detection and removal algorithm (*process*) was written by Dr. Dan Merfeld. Significant contributions to the code have also been made by Braden McGrath and Wally Kulecz.

```
%ACC_FILT This program differentiates eye angular velocity to yield eye
%       angular acceleration.  This is done for each of three DOF.
%
%       D.M. Merfeld 4/17/89
%
% modified to accumulate acceleration magnitude here, rather than
%  in acc_mag, in order to conserve memory
% D. Balkwill 9/26/90

eval(['load ',nysa_path,'bookkeeping:acc_filter.mat'])

x = vel1;
filtzero
mag_acc = x .* x;
clear x

if (dim >= 2)
          x = vel2;
          filtzero
         mag_acc = mag_acc + x .* x;
  clear x
end

if (dim >= 3)
          x = vel3;
          filtzero
  mag_acc = mag_acc + x .* x;
  clear x
end

mag_acc = sqrt(mag_acc);
eval(['save ',nysa_path,'bookkeeping:mag_acc mag_acc'])

clear A B x
```

```
%backup_data
%
% D. Balkwill  10/27/90
%
% Loads the latest acceleration, velocity and SPV (default
% NysA names and folders) and saves them in run coded files
% in the same folder as the input data. The run code is
% loaded from the bookkeeping folder so that it corresponds
% to the data which is being backed up.

if (exist('nysa_path') ~= 1)
   nysa_path = get_path;
end

%load in user-specified file and variable name formats
file_specs

%get run code and number of dimensions
eval(['load ',nysa_path,'bookkeeping:run_code']);
eval(['load ',nysa_path,'bookkeeping:dim']);

%save magnitude of acceleration, for selection of thresholds
eval(['load ',nysa_path,'bookkeeping:mag_acc']);
eval([Acc_Var,' = mag_acc;']);
eval(['save ',data_path,file_name(Acc_File,run_code),' ',Acc_Var]);

if (dim >= 1)
%save raw and slow phase velocities
    eval(['load ',nysa_path,'bookkeeping:vel1']);
   eval([Vel1_Var,' = vel1;']);
   eval(['save ',data_path,file_name(Vel1_File,run_code),' ',Vel1_Var]);
    eval(['load ',nysa_path,'Data_Out:spv1']);
   eval([SPV1_Var,' = spv1;']);
   eval(['save ',data_path,file_name(SPV1_File,run_code),' ',SPV1_Var]);
   eval(['clear ',Vel1_Var]);
   eval(['clear ',SPV1_Var]);
end

if (dim >= 2)
    eval(['load ',nysa_path,'bookkeeping:vel2']);
   eval([Vel2_Var,' = vel2;']);
   eval(['save ',data_path,file_name(Vel2_File,run_code),' ',Vel2_Var]);
    eval(['load ',nysa_path,'Data_Out:spv2']);
   eval([SPV2_Var,' = spv2;']);
    eval(['save ',data_path,file_name(SPV2_File,run_code),' ',SPV2_Var]);
   eval(['clear ',Vel2_Var]);
   eval(['clear ',SPV2_Var]);
end

if (dim >= 3)
    eval(['load ',nysa_path,'bookkeeping:vel2']);
   eval([Vel3_Var,' = vel3;']);
   eval(['save ',data_path,file_name(Vel3_File,run_code),' ',Vel3_Var]);
    eval(['load ',nysa_path,'Data_Out:spv3']);
```

```
   eval([SPV3_Var,' = spv3;']);
   eval(['save ',data_path,file_name(SPV3_File,run_code),' ',SPV3_Var]);
   eval(['clear ',Vel3_Var]);
   eval(['clear ',SPV3_Var]);
end

%clear out variables to conserve memory space
eval(['clear ',Acc_Var]);

clear_specs
clear dim
clear mag_acc vel1 vel2 vel3 spv1 spv2 spv3
```

```
% calibrate
%
% This script allows the user to obtain a calibration factor
% in degrees/unit. It assumes a three point calibration in
% each direction, although the zero is optional. For each axis,
% the user must specify the angular deviations and select the
% "flat" regions of the trace which correspond to fixations on
% the targets.
%
% D. Balkwill 11/27/90

nysa_path = get_path;
file_specs

eval(['load ',nysa_path,'bookkeeping:vel_filter.mat']);
eval(['load ',nysa_path,'bookkeeping:dim']);
eval(['load ',nysa_path,'bookkeeping:colour']);

code = input('Enter Run Code: ','s');

fprintf('\nCalibrating Axis#1...\n');
cal_file = file_name(Cal1_File,code);
eval(['load ',data_path,cal_file]);
eval(['pos = ',Cal1_Var,';']);
eval(['clear ',Cal1_Var]);
t = (([1:length(pos)] - 1)/sample)';

[scale1,noise1,offset1] = three_point(t,pos,colour);
fprintf('\nAxis#1 scale factor = %6.4f deg/unit\n',scale1);

if (dim >= 2)
  fprintf('\nCalibrating Axis#2...\n');
  cal_file = file_name(Cal2_File,code);
  eval(['load ',data_path,cal_file]);
   eval(['pos = ',Cal2_Var,';']);
  eval(['clear ',Cal2_Var]);
  t = (([1:length(pos)] - 1)/sample)';
  [scale2,noise2,offset2] = three_point(t,pos,colour);
  fprintf('\nAxis#2 scale factor = %6.4f deg/unit\n',scale2);
end

if (dim >= 3)
  fprintf('\nCalibrating Axis#3...\n');
  cal_file = file_name(Cal3_File,code);
  eval(['load ',data_path,cal_file]);
   eval(['pos = ',Cal3_Var,';']);
  eval(['clear ',Cal3_Var]);
  t = (([1:length(pos)] - 1)/sample)';
  [scale3,noise3,offset3] = three_point(t,pos,colour);
  fprintf('\nAxis#3 scale factor = %6.4f deg/unit\n',scale3);
end

if (dim == 1)
eval(['save ',nysa_path,'bookkeeping:cal scale1 offset1']);
```

```
elseif (dim == 2)
eval(['save ',nysa_path,'bookkeeping:cal scale1 offset1 scale2 offset2']);
elseif (dim == 3)
eval(['save ',nysa_path,'bookkeeping:cal scale1 offset1 scale2 offset2 scale3 offset3']);
end

clear_specs

clear nysa_path code A B sample dim t pos cal_file
clear scale1 noise1 offset1 scale2 noise2 offset2 scale3 noise3 offset3
```

```
% clear_specs
%
% D.Balkwill 7/1/91
%
% This simply clears out all of the file_specs declarations,
% except for the data_path.  This minimizes programmer's
% anacin consumption.

clear Pos1_Var Pos2_Var Pos3_Var Pos1_File Pos2_File Pos3_File
clear Cal1_Var Cal2_Var Cal3_Var Cal1_File Cal2_File Cal3_File
clear Vel1_Var Vel2_Var Vel3_Var Vel1_File Vel2_File Vel3_File
clear SPV1_Var SPV2_Var SPV3_Var SPV1_File SPV2_File SPV3_File
clear Edited1_Var Edited2_Var Edited3_Var Acc_Var
clear Edited1_File Edited2_File Edited3_File Acc_File
clear Tach_File Tach_Var Parms_File Parms_Var
clear OS1_File OS2_File OS1_Var OS2_Var
clear AVel1_File AVel2_File AVel1_Var AVel2_Var
```

```
% condition
%
% This script conditions the input signal.  It is called from
% 'scale_data' to perform any additional desired filtering on
% the eye position data.
% D. Balkwill 10/26/90

% It currently performs order statistic filtering on the position
% signal with two passes of a combination of 6 and 10 point
% linear filters.

function y = condition(x)
y = x;
y = OS_lin2(y,6,10);
y = OS_lin2(y,6,10);
return;
```

```
%
%DeleteRow -- W. Kulecz
%

function OutMat = DeleteRow(N,InMat)
% return as output matrix, input matrix with row N removed.
[m,n]=size(InMat);
if N==1
  OutMat=InMat(2:m,:);
elseif N==m
  OutMat=InMat(1:m-1,:);
else
  OutMat=[InMat(1:N-1,:) ; InMat(N+1:m,:)];
end
```

```
% diff_list
%

function list = diff_list(x1,x2)

% This functions returns a list of ordered pairs, indicating the
% sections in which the vectors x1 and x2 are unequal.  Each
% ordered pair (x,y) corresponds to a segment in which the points
% indexed by 'x' and 'y' are equal in both vectors, but all points
% in between are unequal.
%
%  D. Balkwill   10/25/90

list = [];
l1 = length(x1);
l2 = length(x2);
l = min([l1 l2]);
d = x1(1:l) - x2(1:l);

i = 1;
while (d(i) == 0)
   i = i + 1;
   if (i >= l)
      break;
   end
end

while (i < l)
   start = i - 1;
   while (d(i) ~= 0)
      i = i + 1;
      if (i >= l)
         break;
      end
   end
   finish = i;
   list = [list ; [start finish]];
   while (d(i) == 0)
      i = i + 1;
      if (i >= l)
         break;
      end
   end
end

clear i l l1 l2 start finish d
return;
```

```
% drift_calibrate
%
% This script allows the user to obtain a calibration factor
% in degrees/unit. It assumes a three point calibration in
% each direction, although the zero is optional. For each axis,
% the user must specify the angular deviations and select the
% "flat" regions of the trace which correspond to fixations on
% the targets.
%
% D. Balkwill 11/27/90
%
% Modified 5/21/91 from 'calibrate' script to perform a first-order
% curve fit to the selected regions, to attempt to compensate
% for drifting EOG signals.

nysa_path = get_path;
file_specs

eval(['load ',nysa_path,'bookkeeping:vel_filter.mat']);
eval(['load ',nysa_path,'bookkeeping:dim']);
eval(['load ',nysa_path,'bookkeeping:colour']);

code = input('Enter Run Code: ','s');

fprintf('\nCalibrating Axis#1...\n');
cal_file = file_name(Cal1_File,code);
eval(['load ',data_path,cal_file]);
eval(['pos = ',Cal1_Var,';']);
eval(['clear ',Cal1_Var]);
t = (([1:length(pos)] - 1)/sample)';

[scale1,noise1,offset1] = drift_three_point(t,pos,colour);
fprintf('\nAxis#1 scale factor = %6.4f deg/unit\n',scale1);

if (dim >= 2)
  fprintf('\nCalibrating Axis#2...\n');
  cal_file = file_name(Cal2_File,code);
  eval(['load ',data_path,cal_file]);
   eval(['pos = ',Cal2_Var,';']);
  eval(['clear ',Cal2_Var]);
  t = (([1:length(pos)] - 1)/sample)';
  [scale2,noise2,offset2] = drift_three_point(t,pos,colour);
  fprintf('\nAxis#2 scale factor = %6.4f deg/unit\n',scale2);
end

if (dim >= 3)
  fprintf('\nCalibrating Axis#3...\n');
  cal_file = file_name(Cal3_File,code);
  eval(['load ',data_path,cal_file]);
   eval(['pos = ',Cal3_Var,';']);
  eval(['clear ',Cal3_Var]);
  t = (([1:length(pos)] - 1)/sample)';
  [scale3,noise3,offset3] = drift_three_point(t,pos,colour);
  fprintf('\nAxis#3 scale factor = %6.4f deg/unit\n',scale3);
```

```
end

if (dim == 1)
eval(['save ',nysa_path,'bookkeeping:cal scale1 offset1']);
elseif (dim == 2)
eval(['save ',nysa_path,'bookkeeping:cal scale1 offset1 scale2 offset2']);
elseif (dim == 3)
eval(['save ',nysa_path,'bookkeeping:cal scale1 offset1 scale2 offset2 scale3 offset3']);
end

clear_specs

clear nysa_path code A B sample dim t pos eogh eogv eogt cal_file
clear scale1 noise1 offset1 scale2 noise2 offset2 scale3 noise3 offset3
```

```
function [scale,noise,offset] = drift_three_point(t,pos,colour)

% This script inputs the angular deviations from the user, and
% calls the 'pick_regions' script so that the fixation regions
% can be selected. For each deviation, the average value over
% all of the regions is calculated. The calibration factor for
% each axis is calculated as the angular difference between the
% positive and negative deviations (in degrees), divided by the
% difference between mean positive and negative deviations (in
% arbitrary units). The calculated mean values are displayed
% to the user graphically as dotted lines, for inspection.
%
% If a zero fixation point is specified, it is used to
% determine the offset value.
%
% The noise is estimated by taking the root-mean-square value of
% the fluctuations about all selected regions.
%
% D. Balkwill 11/27/90
%
% Modified 5/21/91 from the three_point script to perform a
% first-order curve fit to the selected regions, to attempt to
% compensate for drifting EOG signals. This script calls lin_fit
% to determine the slope and intercept of both positive and
% negative regions. Then, these functions are extrapolated to
% a common point, midway between the positive and negative regions.

pos_deg = input('Specify eye movement in degrees, positive trace deflection: ');
if isempty(pos_deg)
  disp('  Default calibration target assumed to be 10 degrees.');
  pos_deg = 10;
end
neg_deg = input('Eye displacement for negative trace deflection: ');
if isempty(neg_deg)
  disp('  Default calibration assumed -10 degrees.');
  neg_deg = -10;
end
if (pos_deg == neg_deg)
  disp('Calibration range cannot be zero, Symmetrical calibration assumed.');
  neg_deg = -1 * pos_deg;
end

% select positive trace deflection regions
disp('');
disp('Use mouse to select flat-top regions of positive trace deflection.');
pos_regions = pick_regions(t,pos,colour);
[m,n] = size(pos_regions);
if (m == 0)
   error('No positive trace deflection flat-top regions, Cannot calibrate.');
else
   pos_samples = [];
   for i=1:m
      pos_samples = [ pos_samples , pos_regions(i,1):pos_regions(i,2) ];
   end
```

```
   pos_anchor = (min(pos_samples) + max(pos_samples)) / 2;
   [pos_slope,pos_int] = lin_fit(t(pos_samples),pos(pos_samples));
   x1 = pos_slope * t(1) + pos_int;
   x2 = pos_slope * t(length(t)) + pos_int;
   hold on
   if (colour == 'y')
      plot([t(1),t(length(t))],[x1,x2],'r:');
   else
      plot([t(1),t(length(t))],[x1,x2],':');
   end
   hold off
end

% select negative trace deflection regions
disp('Now select flat-top regions of negative trace deflection.');
neg_regions = pick_regions(t,pos,colour);
[m,n] = size(neg_regions);
if (m == 0)
   error('Cannot calibrate, no negative trace deflections selected.');
else
   neg_samples = [];
   for i=1:m
      neg_samples = [ neg_samples , neg_regions(i,1):neg_regions(i,2) ];
   end
   neg_anchor = (min(neg_samples) + max(neg_samples)) / 2;
   [neg_slope,neg_int] = lin_fit(t(neg_samples),pos(neg_samples));
   x1 = neg_slope * t(1) + neg_int;
   x2 = neg_slope * t(length(t)) + neg_int;
   hold on
   if (colour == 'y')
      plot([t(1),t(length(t))],[x1,x2],'r:');
   else
      plot([t(1),t(length(t))],[x1,x2],':');
   end
   hold off
end

% calculate scale factor in deg/unit
anchor = round((pos_anchor + neg_anchor) / 2);
pos_val = pos_slope * t(anchor) + pos_int;
neg_val = neg_slope * t(anchor) + neg_int;
scale = (pos_deg - neg_deg) ./ (pos_val - neg_val);

% select optional zero trace deflection regions
y=input('Is there a zero degree calibration target? (y,n) [default = n]  ','s');
if isempty(y)
   y='n';
end
xzero = [];
if y=='y' | y=='Y'
   disp('Select regions in which subject fixated on zero reference.');
   zero_regions = pick_regions(t,pos,colour);
   [m,n] = size(zero_regions);
   if (m == 0)
```

```
      offset = 0;
   else
      for i=1:m
         xzero = [xzero ; pos(zero_regions(i,1):zero_regions(i,2))];
      end
      offset = mean(xzero);
      hold on
       if (colour == 'y')
          plot([t(1),t(length(t))],[offset,offset],'r:');
      else
          plot([t(1),t(length(t))],[offset,offset],':');
      end
      hold off
   end
else
   offset = 0;
end

% display positive and negative mean values
hold on
if (colour == 'y')
   plot([t(1),t(length(t))],[neg_val,neg_val],'r:');
   plot([t(1),t(length(t))],[pos_val,pos_val],'r:');
else
   plot([t(1),t(length(t))],[neg_val,neg_val],':');
   plot([t(1),t(length(t))],[pos_val,pos_val],':');
end
hold off

% estimate noise, in rms degrees
xpos = pos(pos_samples) - ( pos_slope * t(pos_samples) + pos_int );
xneg = pos(neg_samples) - ( neg_slope * t(neg_samples) + neg_int );
if (isempty(xzero) == 0)
   xzero = xzero - mean(xzero);
end
x = [xpos ; xneg ; xzero ];
noise = scale * sqrt(mean(x.*x));

clear pos_regions neg_regions zero_regions pos_samples neg_samples
clear pos_slope pos_int neg_slope neg_int x1 x2
clear anchor pos_anchor neg_anchor pos_val neg_val
clear xpos xneg xzero x
clear i m n y neg_deg pos_deg
return;
```

```
% edit_alg
%

function edited_spv = edit_alg(t,spv,vel,pos,colour)

%   This is the main algorithm for the manual editing of
% slow phase velocity profiles.
%    sample = sampling rate in Hz
%    spv    = slow phase eye velocity vector
%    vel    = raw eye velocity vector
%    pos    = eye position vector
%    colour = flag for colour monitor
%
% The user now has the capability of over-riding faulty
% interpolations made by the detection process. The 'diff_list'
% script is called to return a list of regions over which the
% raw velocity and slow phase velocity differ. The format of
% this list is identical to that of 'flag' in the 'heart' script.
%
% If one wishes to re-edit a previously edited SPV profile, then
% the first line can be deleted so that the 'diffs' list contains
% the differences between raw and *edited* SPV profiles.
%
% If the SPV profile is completely different from the raw
% velocity (due to low-pass or order-statistic filtering for
% instance), then the call to 'diff_list' should be removed.
%
% written by D. Balkwill -- 11/27/90
% some portions ruthlessly and shamelessly stripped from
%    scripts by B. McGrath and W. Kulecz

edited_spv = spv;    % don't overwrite spv
diffs = diff_list(vel,edited_spv);
num_diffs = length(diffs);
highs = [];
num_highs = 0;   % number of regions highlighted
interps = [];
spv_interps = [];
num_interps = 0;   % number of regions interpolated

l = length(edited_spv);
sample = round(1/(t(2) - t(1)));   % assumes t is periodic

% minimum window height to prevent graph from being dominated by noise
rms = sqrt(sum(edited_spv.*edited_spv)/l);
min_height = 3 * rms;

key = 0;
FINISHED = 27;     % escape
PAN_LEFT = 28;     % left arrow
PAN_RIGHT = 29;    % right arrow
SCROLL_LEFT = 11;  % page down
SCROLL_RIGHT = 12; % page up
ACCEPT = 13;       % carriage return
```

```
DELETE_1 = 8;     % backspace
DELETE_2 = 127;   % delete
ZOOM_IN = 30;     % up arrow
ZOOM_OUT = 31;    % down arrow
FAST_ZOOM_IN = 46;  % decimal
FAST_ZOOM_OUT = 48;  % zero
COMPLETE_PLOT_1 = 97; % 'a' key
COMPLETE_PLOT_2 = 65; % 'A' key
% note: 1, 2, and 3 are reserved for mouse button(s)

num_pick = 0;   % number of points picked
os = 1;         % offset of start of current trace, in samples
w = l - 1;      % width of trace, in samples
redraw = 1;     % flag for plotting
mf = 1;         % magnification factor
mag_thresh = l / (sample * 10);  % 10 seconds

while (key ~= FINISHED)

   if (redraw == 1)
      df = floor(w/2000);
      if (df < 1)
         df = 1;
      end
      er = edited_spv(os:df:os+w);
      tr = t(os:df:os+w);
      vr = vel(os:df:os+w);
      pr = pos(os:df:os+w);

% leave some blank space above and below trace for aesthetics
      mxv = max(er);
      if (mxv < 0)
         mx = mxv * 0.9;
      else
         mx = mxv * 1.1;
      end
      mnv = min(er);
      if (mnv < 0)
         mn = mnv * 1.1;
      else
         mn = mnv * 0.9;
      end
      old = mx - mn;
      if (old < min_height)
         mx = mx + (min_height - old)/2;
         mn = mx - min_height;
      end

% ensure that position data appears on plot
      pr = pr - min(pr) + mnv;

      hold off
      axis([tr(1) tr(length(tr)) mn mx]);
       if (colour == 'y')
```

```
   if (mf < mag_thresh)
       plot(tr,er,'w')
        text(.4,0,'black = SPV','sc')
   else
          plot(tr,vr,'r-',tr,pr,'g-',tr,er,'w')
        text(.12,0,'black = SPV','sc')
        text(.4,0,'red = raw velocity','sc')
        text(.7,0,'green = eye position','sc')
   end

   % plot highlighted regions in green, solid
   hold on
   for i=1:num_highs
      x3 = highs(i,1);
      x4 = highs(i,2);
       plot([t(x3),t(x3)],[mn,mx],'g')
       plot([t(x4),t(x4)],[mn,mx],'g')
       plot([t(x3),t(x4)],[mn,mx],'g')
   end

   % plot picked regions in blue, dash-dotted
   for i=1:num_interps
      x3 = interps(i,1);
      x4 = interps(i,2);
       plot([t(x3),t(x3)],[mn,mx],'b-.')
       plot([t(x4),t(x4)],[mn,mx],'b-.')
       plot([t(x3),t(x4)],[mn,mx],'b-.')
      plot(t(x3:x4),spv_interps(1:(x4-x3+1),i),'b-')
   end
   % plot currently picked point in blue, dotted
   if (num_pick == 1)
       plot([t1,t1],[mn,mx],'b:')
   end
   hold off

else

   if (mf < mag_thresh)
      plot(tr,er)
        text(.4,0,'solid = SPV','sc')
   else
        plot(tr,vr,':',tr,pr,'--',tr,er)
        text(.12,0,'solid = SPV','sc')
       text(.4,0,'dotted = raw velocity','sc')
       text(.7,0,'dashed = eye position','sc')
   end

   % plot highlighted regions in dashed
   hold on
   for i=1:num_highs
      x3 = highs(i,1);
      x4 = highs(i,2);
       plot([t(x3),t(x3)],[mn,mx],'--')
```

```
            plot([t(x4),t(x4)],[mn,mx],'--')
            plot([t(x3),t(x4)],[mn,mx],'--')
        end

        % plot picked regions in dash-dotted
        for i=1:num_interps
           x3 = interps(i,1);
           x4 = interps(i,2);
            plot([t(x3),t(x3)],[mn,mx],'-.')
            plot([t(x4),t(x4)],[mn,mx],'-.')
            plot([t(x3),t(x4)],[mn,mx],'-.')
            plot(t(x3:x4),spv_interps(1:(x4-x3+1),i),'-.')
        end
        % plot currently picked point in dotted
        if (num_pick == 1)
            plot([t1,t1],[mn,mx],'-.')
        end
        hold off
    end

     text(.7,.93,['magnification = ',int2str(round(mf)),' X'],'sc')
    redraw = 0;

end

[x,y,key] = ginput(1);

if (key == ZOOM_IN)     % increase magnification factor

    old=mf;
    mf=min(old*2,max(old,floor(l/100)));
    if mf==old    % maximum magnification of 100X
       redraw=0;
    else
       redraw=1;
       w=floor(l/mf);
    end

elseif (key == FAST_ZOOM_IN)  % fast two-point zoom

    % first point of region to zoom into
    [t3,y,key] = ginput(1);
    if ((key ~= DELETE_1) & (key ~= DELETE_2))

        % bounds check on first point of region
        if (t3 < tr(1))
           t3 = tr(1);
        elseif (t3 > tr(length(tr)))
           t3 = tr(length(tr));
        end
        x3 = 1 + round(t3 * sample);
        t3 = (x3 - 1)/sample;

        % display first point
```

```
      hold on
       if (colour == 'y')
          plot([t3,t3],[mn,mx],'r:');
      else
          plot([t3,t3],[mn,mx],':');
      end
      hold off
      redraw = 1;

      % second point of region to zoom into
      [t4,y,key] = ginput(1);

      % allow user to abort zoom via delete key
      if ((key ~= DELETE_1) & (key ~= DELETE_2))

         % bounds check on second point of region
         if (t4 < tr(1))
            t4 = tr(1);
         elseif (t4 > tr(length(tr)))
            t4 = tr(length(tr));
         end
         x4 = 1 + round(t4 * sample);
         t4 = (x4 - 1)/sample;

         % display second point
         hold on
          if (colour == 'y')
             plot([t4,t4],[mn,mx],'r:');
         else
             plot([t4,t4],[mn,mx],':');
         end
         hold off

         % swap order of points if needed
         if (x4 < x3)
            old = x4;
            x4 = x3;
            x3 = old;
         end

         % calculate new magnification parameters
         if (x3 ~= x4)
            os = x3;
            w = x4 - x3;
            mf = 1/w;
         end
      end
   end

elseif (key==ZOOM_OUT)  % decrease magnification

   if (mf == 1)  % already completely zoomed out
      redraw = 0;
   else
```

```
      redraw=1;
      old=mf;
      mf=max(floor(old/2),1);
      w=floor(l/mf);
      if (w >= l)
         w = l - 1;
      end
      if ((os+w)>l)
        os=floor(max(1,l-w));
      end
    end

  elseif ((key == COMPLETE_PLOT_1) | (key == COMPLETE_PLOT_2) | (key ==
FAST_ZOOM_OUT))   % display entire plot

    os = 1;
    mf = 1;
    w = l - 1;
    redraw = 1;

  elseif (key==PAN_RIGHT)   % increase offset by quarter-screen

    old=os;
    os=floor(max(1,min(l-w,os+0.25*w)));
    if old==os      % already panned to end
      redraw=0;
    else
      redraw=1;
    end

  elseif (key==PAN_LEFT)   % decrease offset by quarter-screen

    old=os;
    os=floor(max(1,os-0.25*w));
    if os==old      % already panned to beginning
      redraw=0;
    else
      redraw=1;
    end

  elseif (key==SCROLL_RIGHT)   % jump display one screenful right

    old=os;
    os=floor(max(1,min(os+w,l-w)));
    if os==old      % already panned to end
      redraw=0;
    else
      redraw=1;
    end

  elseif (key==SCROLL_LEFT)   % jump display one screenful left

    old=os;
    os=floor(max(1,os-w));
```

```
  if old==os      % already panned to beginning
    redraw=0;
  else
    redraw=1;
  end

elseif (key==ACCEPT)  % accept fast phase interpolations

  if (num_pick == 0)
    for i=1:num_interps

      % substitute new values into edited SPV
      x1 = interps(i,1);
      x2 = interps(i,2);
      edited_spv(x1:x2) = spv_interps(1:(x2-x1+1),i);

      % add region to list of differences
      if (x2 < diffs(1,1))  % add to beginning
        diffs = [[x1 x2] ; diffs];
      elseif (x1 > diffs(num_diffs,2))  % add to end
        diffs = [diffs ; [x1 x2]];
      else
        for j=1:num_diffs
          if (diffs(j,1) > x2)
            break;
          end
        end
        diffs = [diffs(1:j-1,:) ; [x1 x2] ; diffs(j:num_diffs,:)];
      end
      num_diffs = num_diffs + 1;

    end

    % reset appropriate values
    num_interps = 0;
    clear interps spv_interps
    interps = [];
    spv_interps = [];
    num_pick = 0;
    redraw = 1;
  end

elseif ((key==DELETE_1) | (key==DELETE_2))

  if (num_pick > 0)  % wipe out currently picked point
    num_pick = 0;
    redraw = 1;
  elseif (num_highs > 0)  % wipe out highlit regions
    for i=1:num_highs
      x3 = highs(i,1);
      x4 = highs(i,2);
      % copy raw velocity values back in
      edited_spv(x3:x4) = vel(x3:x4);
      index = InList(x3,diffs);
```

```
      diffs = DeleteRow(index,diffs);
    end
    num_diffs = num_diffs - num_highs;
    num_highs = 0;
    clear highs
    highs = [];
    redraw = 1;
  elseif (num_interps > 0)  % wipe out last interpolation
    redraw = 1;
    num_interps = num_interps - 1;
    interps = interps(1:num_interps,:);
    spv_interps = spv_interps(:,1:num_interps);
  end

elseif (key==1) | (key==2) | (key==3)  % up to three-button mouse input

  if (num_pick == 0)  % this is the first picked point

    % bounds check on picked point
    if (x < tr(1))
      x = tr(1);
    elseif (x > tr(length(tr)))
      x = tr(length(tr));
    end

    % convert time value to sample number
    x1 = 1 + round(x * sample);

    % see if point is in a selected region
    index1 = InList(x1,diffs);
    if (index1 > 0)
      index2 = InList(x1,highs);
      if (index2 > 0)  % de-highlight region
        num_highs = num_highs - 1;
        highs = DeleteRow(index2,highs);
        redraw = 1;
      else  % highlight region for future deletion
        x1 = diffs(index1,1);
        x2 = diffs(index1,2);
        t1 = (x1 - 1)/sample;
        t2 = (x2 - 1)/sample;
        num_highs = num_highs + 1;
        highs(num_highs,:) = [x1 x2];
        if (colour == 'y')
          hold on
          plot([t1,t1],[mn,mx],'g');
          plot([t2,t2],[mn,mx],'g');
          plot([t1,t2],[mn,mx],'g');
        else
          hold on
          plot([t1,t1],[mn,mx],'-');
          plot([t2,t2],[mn,mx],'-');
          plot([t1,t2],[mn,mx],'-');
        end
```

```
    end
  else   % point is not already in a selected region
    % display as first point of region being selected
    t1 = (x1 - 1)/sample;
    num_pick = 1;
    hold on
     if (colour == 'y')
        plot([t1,t1],[mn,mx],'b:');
    else
        plot([t1,t1],[mn,mx],'-.');
    end
    hold off
  end
elseif (num_pick == 1)  % second picked point

  % bounds check on picked point
  if (x < tr(1))
    x = tr(1);
  elseif (x > tr(length(tr)))
    x = tr(length(tr));
  end

  % convert time value to sample number
  x2 = 1 + round(x * sample);
  t2 = (x2 - 1)/sample;

  if (x2 == x1)  % cannot have interval of zero width
    num_pick = 0;
    redraw = 1;
  else
    if (x2 < x1)   % order picked points
      old = x1;
      x1 = x2;
      x2 = old;
    end
    num_pick = 0;
    hold on
     if (colour == 'y')
        plot([t2,t2],[mn,mx],'b:');
    else
        plot([t2,t2],[mn,mx],'-.');
    end

    % interpolate linearly across picked interval
    pick_spv = edited_spv(x1:x2);
    pick_l = length(pick_spv);
    pick_t = t(x1:x2);
    slope = (pick_spv(pick_l) - pick_spv(1)) / (pick_l - 1);
    j = pick_l - 1;
    for i=2:j
      pick_spv(i) = pick_spv(1) + ((i-1) * slope);
    end
     if (colour == 'y')
        plot(pick_t,pick_spv,'b-');
```

```
            else
                plot(pick_t,pick_spv,'-.');
            end
            hold off

            % add region to list of interpolated regions, and
            % add interpolated vector to its list, padding the
            % list of interpolated vectors with zeros as needed
            num_interps = num_interps + 1;
            interps = [interps ; [x1 x2]];
            [mm,nn] = size(spv_interps);
            if (pick_l < mm)
               pick_spv = [pick_spv ; zeros(mm-pick_l,1)];
            elseif (pick_l > mm)
               if (nn > 0)
                  spv_interps = [spv_interps ; zeros(pick_l-mm,nn)];
               end
            end
            spv_interps = [spv_interps pick_spv];
         end
      end
   end
end
return;
```

```
%edit_spv

% This script is used for manual editing of a slow phase
% velocity profile. The user specfies a run code and an axis
% number, and the corresponding eye position, velocity, and
% slow phase velocity are loaded from the data_path. The eye
% position is then corrected to degrees (relative, not absolute)
% so that all displayed data will be in deg or deg/sec.
%
% Modificatio          run was the last one processed, as told
% by the user          is loaded from the nysa_path. This is
% faster becau.        osition does not have to be corrected
% to degrees.
%
% The 'edit_alg' script is then called to perform the actual
% editing process -- this is simply the interface between the
% basic algorithm and the NysA format.
%
% Finally, the user is giving the option of saving the edited
% SPV profile, or aborting so as not to overwrite previously
% saved data.
%
% D. Balkwill -- 11/27/90

if (exist('nysa_path') ~= 1)
   nysa_path = get_path;
end

file_specs

eval(['load ',nysa_path,'bookkeeping:vel_filter.mat']);
eval(['load ',nysa_path,'bookkeeping:dim']);
eval(['load ',nysa_path,'bookkeeping:colour']);

run_code = input('Enter Run Code: ','s');

latest = input('Was this the last processed run? (y/n) [default = y]  ','s');
if (isempty(latest))
   latest = 'y';
end

dn = 1;
accept_flag = 0;
while (accept_flag == 0)
   ans = input(['Enter number of axis to edit [default = ',int2str(dn),']  ']);
   if (isempty(ans) == 0)
      dn = ans;
   end
   if (dn <= dim)
      accept_flag = 1;
   else
      fprintf(['Axis number cannot exceed ',int2str(dim),'.\n']);
      dn = 1;
   end
```

```
end
dn = int2str(dn);

if ((latest == 'y') | (latest == 'Y'))

    eval(['load ',nysa_path,'Data_In:pos',dn]);
    eval(['pos = pos',dn,';']);
    eval(['clear pos',dn]);
    eval(['load ',nysa_path,'bookkeeping:vel',dn]);
    eval(['vel = vel',dn,';']);
    eval(['clear vel',dn]);
    eval(['load ',nysa_path,'Data_Out:spv',dn]);
    eval(['spv = spv',dn,';']);
    eval(['clear spv',dn]);

else

    if (dn == '1')
      in_file = file_name(Pos1_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['pos = ',Pos1_Var,';']);
      eval(['clear ',Pos1_Var]);
      clear in_file
      in_file = file_name(Vel1_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['vel = ',Vel1_Var,';']);
      eval(['clear ',Vel1_Var]);
      clear in_file
      in_file = file_name(SPV1_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['spv = ',SPV1_Var,';']);
      eval(['clear ',SPV1_Var]);
      clear in_file
    elseif (dn == '2')
      in_file = file_name(Pos2_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['pos = ',Pos2_Var,';']);
      eval(['clear ',Pos2_Var]);
      clear in_file
      in_file = file_name(Vel2_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['vel = ',Vel2_Var,';']);
      eval(['clear ',Vel2_Var]);
      clear in_file
      in_file = file_name(SPV2_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['spv = ',SPV2_Var,';']);
      eval(['clear ',SPV2_Var]);
      clear in_file
    elseif (dn == '3')
      in_file = file_name(Pos3_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['pos = ',Pos3_Var,';']);
      eval(['clear ',Pos3_Var]);
```

```
      clear in_file
      in_file = file_name(Vel3_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['vel = ',Vel3_Var,';']);
      eval(['clear ',Vel3_Var]);
      clear in_file
      in_file = file_name(SPV3_File,run_code);
      eval(['load ',data_path,in_file]);
       eval(['spv = ',SPV3_Var,';']);
      eval(['clear ',SPV3_Var]);
      clear in_file
   end
   x = pos;
   filtzero;
   cal_factor = (max(vel) - min(vel)) / (max(x) - min(x));
   cal_factor = (cal_factor + std(vel)/std(x))/2;
   pos = pos * cal_factor;
   clear x cal_factor

end

l = length(spv);
t = [0:(l-1)]/sample;

edited_spv = edit_alg(t,spv,vel,pos,colour);

yn = input('Do you wish to save this edition?  (y/n)  [default = y]','s');
if (isempty(yn))
   yn = 'y';
end

if ((yn ~= 'n') & (yn ~= 'N'))

   axis([1 2 3 4]);axis;
   plot(t,edited_spv);

   xlabel('Time (sec)');
   ylabel('Slow Phase Velocity (deg/sec)');

    eval(['title("Edited SPV -- Axis#',dn,' -- ',run_code,'")']);
    var = ['Edited',dn,'_Var'];
   eval([var,'= edited_spv;']);
   eval(['in_file = file_name(Edited',dn,'_File,run_code);']);
    eval(['save ',data_path,in_file,' ',var]);
    eval(['save ',nysa_path,'Data_Out:edited_spv',dn,' ',var]);
end

eval(['clear ',var]);
clear_specs

clear nysa_path dn dim pos vel spv edited_spv t sample run_code A B
clear accept_flag ans var mn mx l data_path latest in_file colour
```

```
%file_name

%  D. Balkwill  9/26/90
%
% returns the input file name, given the file spec and run code

function in_file = file_name(file_spec,code)

in_file = file_spec;
l = length(in_file);

i = 1;
while (i <= l)
   if (in_file(i) == '#')
      break;
   end
   i = i + 1;
end

if (i == l)
   in_file = [in_file(1:l-1),code];
elseif (i < l)
   in_file = [in_file(1:i-1),code,in_file(i+1:l)];
end

% make sure there are no blank spaces in the file name
l = length(in_file);
i = 1;
while (i <= l)
   if (in_file(i) == ' ')
      in_file = [in_file(1:i-1),in_file(i+1:l)];
      l = l - 1;
   else
      i = i + 1;
   end
end

clear i l
return;
```

```
%file_specs
%   D. Balkwill 7/1/91
%
% The variables in this file (left-hand side of = sign) are
%   used by the scripts to give the user control over the names
%   of both variables and files in which the data are stored.
% The right-hand side is to be filled in by the user.
% The # sign is a place-holder for a run code to distinguish
%   between different files; whereas, the remainder of the
%   filename is constant for all run codes.
% Variable names are to be constant for all run codes.
% File and variable names are placed within single quotes

data_path = 'Analysis:';

Pos1_File = '#.eogh';
Pos2_File = '#.eogv';
Cal1_File = '#.eogh';
Cal2_File = '#.eogv';
OS1_File = '#.osh';
OS2_File = '#.osv';
Vel1_File = '#.velh';
Vel2_File = '#.velv';
SPV1_File = '#.spvh';
SPV2_File = '#.spvv';
Edited1_File = '#.editedh';
Edited2_File = '#.editedv';
AVel1_File = '#.aspvh';
AVel2_File = '#.aspvv';
Acc_File = '#.acc';
Tach_File = '#.tach';
Parms_File = '#.parms';

Pos1_Var = 'eogh';
Pos2_Var = 'eogv';
Cal1_Var = 'eogh';
Cal2_Var = 'eogv';
OS1_Var = 'osh';
OS2_Var = 'osv';
Vel1_Var = 'velh';
Vel2_Var = 'velv';
SPV1_Var = 'spvh';
SPV2_Var = 'spvv';
Edited1_Var = 'editedh';
Edited2_Var = 'editedv';
AVel1_Var = 'aspvh';
AVel2_Var = 'aspvv';
Acc_Var = 'acc';
Tach_Var = 'tach';

%Ensure that colon is last character in path name, for Mac convention
if (data_path(length(data_path)) ~= ':')
   data_path = [data_path,':'];
end
```

```
% fill
%
%       This subroutine estimates the slow phase velocity during a fast phase
%       to be equal to the slow phase velocity just before the saccade. This
%       estimate of the slow phase velocity is used to fill in the duration of
%       the fast phase. The median of the previous five velocity points is
% used as the interpolated value.
%       D.M. Merfeld 3/30/89
%
%Modified by D. Balkwill 9/26/90
% If the start of the first detected event is in the first five points,
% the median of the five velocity points immediately following the
% event is used as the interpolated value. If there are neither five
% points before nor after the event, the median value of all of the
% points (excluding the event) is used as the interpolated value.
%NOTE: velocity vector is assumed to be in variable x

%get number of detected events
[mm,nn] = size(flag);
number = mm;

l = length(x);
n = 1;
while (n <= number)
        start = flag(n,1);
          finish = flag(n,2);
   % new velocity to be interpolated is median value
   if (start < 5)  % start is too early
      if ((finish+4) <= l)
         v_beg = median(x(finish:finish+4));
      else  % finish is too late -- do best possible
         v_beg = median([x(1:start)' x(finish:l)']);
      end
   else
            v_beg = median(x(start-4:start));
   end
   x(start+1:finish-1) = v_beg * ones(finish-start-1,1);
         n = n + 1;
end

clear number n start finish mm nn v_beg l
```

```
%FILTZERO Performs zero phase shift filtering for FIR filters (ONLY!)
%       by padding signal to be filtered with non-zero values at
%       beginning and end of data sequence.
%       Called with the same parameters and order as Matlab filter command.

%       D.M. Merfeld 3/30/89

nx=max(size(x));  %this section of code calculates the eye velocity
nB=max(size(B));
x(nx+1:nx+((nB-1)/2))=x(nx).*ones(((nB-1)/2),1);
[temp,z]=filter(B,A,x(1).*ones(((nB-1)/2),1));
clear temp
x=filter(B,A,x,z);
x=x(((nB-1)/2+1):nx+((nB-1)/2));
clear z nx nB
```

```
% get_path
%
% Written by: D. Balkwill  9/26/90
%
% This routine searches the MatLab path specification for a file named
%    nysa_path, which is to contain a variable named 'nysa_path'.

function nysa_path = get_path

%get MatLab path specification
mat_path = getenv('MATLABPATH');
l = length(mat_path);

skip = 0;
b = 0;
status = 0;
while ((status ~= 2) & (b < l))
   a = b + 1;
   b = a + 1;
   while (mat_path(b) ~= ';')
      if (mat_path(b) == ' ') %skip over paths with blanks because
         skip = 1;          %    they mess up 'exist' statement
         break;
      end
      b = b + 1;
      if (b > l)
         break;
      end
   end
   if (skip == 0)
       status = eval(['exist('',mat_path(a:b-1),':nysa_path'')']);
   else
      skip = 0;  %reset
   end
end

if (status == 2)
    eval(['load ',mat_path(a:b-1),':nysa_path']);
else
   nysa_path = input('Enter NysA folder specification: ','s');
end

l = length(nysa_path);
if (nysa_path(l) ~= ':')
    nysa_path = [nysa_path,':'];
end
if (status ~= 2)
    eval(['save ',nysa_path,'nysa_path nysa_path']);
end

clear a b status mat_path l skip
return;
```

```
% heart
%
% Detects the large fast phases by using two acceleration thresholds
%       set by the user. The subroutine finds the start and finish of fast
%       phases. The variable 'start' marks the last good data point before
%       the fast phase, while the variable 'finish' marks the first good
%  data point after the fast phase.
% An event is detected when the acceleration exceeds 'thresh_acc'.
% The beginning and end of the fast phase occur when a certain number
%  of consecutive points (specified by 'under') have an acceleration
%   below 'thresh_end'.
% In order to speed up execution, the acceleration profile is padded with
%  'dummy' zeroes at the beginning and end of the run. This prevents a
%  fast phase from exceeding the beginning or end of the run, without
%  having to explicitly check every point.
% In order to further speed up execution, a counter is used instead
%  of the 'max' routine so that each point is checked only once
%       D.M. Merfeld 3/30/89
% D. Balkwill 11/27/90

under = 3;  % number of samples to be under thresh_end
dummy = max(under,10);
l = length(mag_acc);
stop = l + dummy;
mag_acc = [zeros(1,dummy),mag_acc',zeros(1,dummy)]';

finish = 0;
event = 0;

m = 1;
while (m <= stop)

   if (mag_acc(m) >= thresh_acc)   %fast phase detected here

%find beginning of event
      b = m;
      count = 0;
      while (count < under)
         b = b - 1;
         if (mag_acc(b) < thresh_end)
            count = count + 1;
         else
            count = 0;
         end
      end

%check for overlapping events
      if (b > finish)
         event = event + 1;
         start = b + 1;  %play with this to optimize detection
      end

%find end of event
      count = 0;
```

```
      while (count < under)
         m = m + 1;
         if (mag_acc(m) < thresh_end)
            count = count + 1;
         else
            count = 0;
         end
      end
      finish = m;    %play with this to optimize detection

%add event to list, removing offset due to padding
                     flag(event,:)=[start-dummy finish-dummy];

   end
   m = m + 1;
end

%ensure that all first and last events are within range of the run
if (event > 0)
   if (flag(1,1) <= 0)
      flag(1,1) = 1;
   end
   if (flag(event,2) > l)
      flag(event,2) = l;
   end
else
   flag = [];
end

%remove padding
mag_acc = mag_acc(dummy+1:l-dummy);

clear under b m stop start finish event dummy l count
```

```
%init

% D.M. Merfeld  6/13/90
% D. Balkwill  10/11/90

%get path specification for location of NysA scripts and files
nysa_path = get_path;
ans = input(['Enter NysA path [default = ',nysa_path,']   '],'s');
if (isempty(ans) == 0)
   nysa_path = ans;
   if (nysa_path(length(nysa_path)) ~= ':')
      nysa_path = [nysa_path,':'];
   end
   eval(['save ',nysa_path,'nysa_path nysa_path'])
end

%get number of dimensions for multi-dimensional algorithm
dim = 1;  % previous value, or 1, is default
if (exist([nysa_path,'bookkeeping:dim']) == 2)
   eval(['load ',nysa_path,'bookkeeping:dim']);
end
accept_flag = 0;

while (accept_flag == 0)
   ans = input(['Enter number of dimensions [default = ',int2str(dim),']  ']);
   if (isempty(ans) == 0)
      dim = ans;
   end
   if ((dim == 1) | (dim == 2) | (dim == 3))
      accept_flag = 1;
   else
      fprintf('Number of dimensions must be 1, 2, or 3.\n');
      dim = 1;
   end
end

eval(['save ',nysa_path,'bookkeeping:dim dim'])
clear dim

%get acceleration thresholds
if (exist([nysa_path,'bookkeeping:thresh.mat']) == 2)
   eval(['load ',nysa_path,'bookkeeping:thresh.mat'])
else
   thresh_acc = -2000;
   thresh_end = -1000;
   eval(['save ',nysa_path,'bookkeeping:thresh.mat'])
end

if ((thresh_acc < 0) | (thresh_end < 0))
   fprintf('\nPeak threshold = <unknown>')
   fprintf('\nEnd threshold = <unknown>')
else
   fprintf(['\nPeak threshold = ',int2str(thresh_acc)])
   fprintf(['\nEnd threshold = ',int2str(thresh_end)])
```

```
end
ans = input('Are these the values you want to use? (y,n) [default = y]  ','s');
if (isempty(ans))
   ans = 'y';
elseif (ans == 'Y')
        ans = 'y';
end

while (ans ~= 'y')

   thresh_acc=input('Enter the value for the peak threshold: ');
   thresh_end=input('Enter the value for the end threshold: ');
   if ((thresh_acc < 0) | (thresh_end < 0))
      fprintf('\nPeak threshold = <unknown>')
      fprintf('\nEnd threshold = <unknown>')
   else
      fprintf(['\nPeak threshold = ',int2str(thresh_acc)]);
      fprintf(['\nEnd threshold = ',int2str(thresh_end)]);
   end
         ans = input('Are these the values you want to use? (y,n) [default = y]  ','s');
   if (isempty(ans))
      ans = 'y';
         elseif (ans == 'Y')
               ans = 'y';
         end
end

eval(['save ',nysa_path,'bookkeeping:thresh.mat thresh_acc thresh_end'])
clear thresh_acc thresh_end

%get sampling rate
sample = 120;      % default is 120 or previous value
if (exist([nysa_path,'bookkeeping:vel_filter.mat']) == 2)
   eval(['load ',nysa_path,'bookkeeping:vel_filter.mat']);
end

accept_flag = 0;
while (accept_flag == 0)

   ans = input(['Enter sampling rate [default = ',int2str(sample),']  ']);
   if (isempty(ans) == 0)
      sample = ans;
   end
   accept_flag = 1;

%% These four lines ensure that sample is stored as a number.
%% MatLab stores it kind of bizarre as a half string (?) otherwise.
   tmp = round(sample);
   clear sample
   sample = tmp;
   clear tmp

   if (sample==60)
            eval(['load ',nysa_path,'filters:vel_filters:vel9_60.mat'])
```

```
            eval(['save ',nysa_path,'bookkeeping:vel_filter.mat A B sample'])
          eval(['load ',nysa_path,'filters:acc_filters:acc3_60.mat'])
          eval(['save ',nysa_path,'bookkeeping:acc_filter.mat A B sample'])
  elseif (sample==100)
          eval(['load ',nysa_path,'filters:vel_filters:vel9_100.mat'])
          eval(['save ',nysa_path,'bookkeeping:vel_filter.mat A B sample'])
            eval(['load ',nysa_path,'filters:acc_filters:acc5_100.mat'])
            eval(['save ',nysa_path,'bookkeeping:acc_filter.mat A B sample'])
  elseif (sample==120)
          eval(['load ',nysa_path,'filters:vel_filters:vel9_120.mat'])
            eval(['save ',nysa_path,'bookkeeping:vel_filter.mat A B sample'])
          eval(['load ',nysa_path,'filters:acc_filters:acc5_120.mat'])
          eval(['save ',nysa_path,'bookkeeping:acc_filter.mat A B sample'])
  elseif (sample==128)
          eval(['load ',nysa_path,'filters:vel_filters:vel9_128.mat'])
            eval(['save ',nysa_path,'bookkeeping:vel_filter.mat A B sample'])
          eval(['load ',nysa_path,'filters:acc_filters:acc5_128.mat'])
          eval(['save ',nysa_path,'bookkeeping:acc_filter.mat A B sample'])
  elseif (sample==200)
            eval(['load ',nysa_path,'filters:vel_filters:vel5_200.mat'])
            eval(['save ',nysa_path,'bookkeeping:vel_filter.mat A B sample'])
            eval(['load ',nysa_path,'filters:acc_filters:acc7_200.mat'])
            eval(['save ',nysa_path,'bookkeeping:acc_filter.mat A B sample'])
  else
    accept_flag = 0;
    fprintf('Sampling rate must be 60, 100, 120, 128, or 200.\n');
    sample = 120;
  end
end

colour = 'y';
if (exist([nysa_path,'bookkeeping:colour']) == 2)
   eval(['load ',nysa_path,'bookkeeping:colour']);
end

ans = input(['Do you have a colour monitor? (y/n) [default = ',colour,']  '],'s');
if (isempty(ans) == 0)
   if ((ans == 'n') | (ans == 'N'))
      colour = 'n';
  else
      colour = 'y';
  end
end

eval(['save ',nysa_path,'bookkeeping:colour colour']);

clear ans A B sample nysa_path accept_flag colour
```

```
%
%InList -- W. Kulecz
%

function K = InList(value,list)
% scan a matrix and return row number, K, such that:
%       list(K,1) <= value <= list(K,2)
% return K = 0 if value not in list.
[m,n]=size(list);
if m==0
  K=0;
  return;
end
if n < 2
  error('List matrix must have at least two columns.');
end
K=m;
while K > 0
  if (value >= list(K,1)) & (value <= list(K,2))
    return;
  end
K=K-1;
end
```

```
%lin_fit
%
% D. Balkwill 5/21/91
% This determines a linear curve fit, y = mt + b, to a segment,
% x, by calculating optimal m and b. The error function to
% minimize is the sum of squares:
%     sum{ [(mt + b) - x]^2 }
% t is the sequence of time values
% x is the sequence of actual values at each time
% y is the sequence of curve fit points
%
% To minimize with respect to m and b, we require the partial
% derivatives to be zero, yielding:
%     sum{ 2[(mt + b) - x]t } = 0       d/dm
%     sum{ 2[(mt + b) - x] }  = 0       d/db
%
% Hence,
%       m * sum{ t^2 } + b * sum{ t } = sum{ xt }
%       m * sum{ t }   + b * sum{ 1 } = sum{ x }
%
% Therefore,
%       m = [ sum{ xt } * sum{ 1 } - sum{ t } * sum{ x }]
%         / [ sum{ t^2 } * sum{ 1 } - sum{ t } * sum{ t }]
%       b = [ sum{ t^2 } * sum{ x } - sum{ xt } * sum{ t }]
%         / [ sum{ t^2 } * sum{ 1 } - sum{ t } * sum{ t }]

function [m,b] = lin_fit(t,x)

[n1,n2] = size(t);
[n3,n4] = size(x);

if (n1 ~= n3)
   t = t';
   [n1,n2] = size(t);
end
l1 = n1 * n2;
l2 = n3 * n4;
if (l1 >= l2)
   l = l2;       % sum{ 1 }
   t = t(1:l);
else
   l = l1;       % sum{ 1 }
   x = x(1:l);
end

A = sum(t .* t);   % sum{ t^2 }
B = sum(t);        % sum{ t }
C = sum(x);        % sum{ x }
D = sum(x .* t);   % sum{ xt }

m = (D * l - B * C) / (A * l - B * B);
b = (A * C - D * B) / (A * l - B * B);

return;
```

```
function y = OS_lin2(x,N1,N2)

% This function performs order statistic filtering (first stage
% of Engelken, 1990) on an input signal x to sharpen corners and
% reduce noise. It uses two windows of length N1 and N2,
% allowing linear root signals. The new value is therefore
% the median of five values.
%
% D. Balkwill 10/28/90

l = length(x);

% calculate window coefficients for forward filters
h1F = PFMH1(N1);
h2F = PFMH1(N2);

% backward filters have same coefficients, but in reverse
h1B = h1F;
for i=1:N1
   h1B(i) = h1F(N1-i+1);
end
h2B = h2F;
for i=1:N2
   h2B(i) = h2F(N2-i+1);
end

% Use Matlab filter command to maximize speed of execution,
% applying forward and backward windows to appropriate range
% of the input signal.
xF1 = filter(h1F,1,x);
xF1 = [x(1:N1)' xF1(N1:l-1)'];
xB1 = filter(h1B,1,x);
xB1 = [xB1(N1+1:l)' x(l-N1+1:l)'];

xF2 = filter(h2F,1,x);
xF2 = [x(1:N2)' xF2(N2:l-1)'];
xB2 = filter(h2B,1,x);
xB2 = [xB2(N2+1:l)' x(l-N2+1:l)'];

y = median([x xF1 xB1 xF2 xB2]')';

clear xB1 xF1 xB2 xF2 i l h1F h1B h2F h2B
return;
```

```
function h1 = PFMH1(N)

% This calculates the coefficients for a linear order statistic
% window of length N.
%
% D. Balkwill  10/28/90

a = (4*N + 2)/(N*(N-1));
b = 6/(N*(N-1));
h1 = a * ones(1,N) - b * [1:N];
clear a b
return;
```

```
% pick_regions
%

function regions = pick_regions(t,pos,colour)

% This is the main algorithm for the manual picking of
% calibration regions
%    t      = time coordinate, equally spaced at sampling period
%    pos    = eye position vector
%    colour = flag for colour monitor
%
% A region is selected by picking its beginning and end with
% the mouse.  A selected region is highlighted by picking within
% that region.  A highlighted region is un-highlighted by
% picking again withing that region.  A selected region is
% de-selected by highlighting it, and then pressing the delete
% key.
%
% The user has complete control over pan and zoom features, as
% well as selection and de-selection of regions.  The plotting
% makes use of different colours and line types, as available.
%
% D. Balkwill  11/27/90

l = length(pos);
sample = round(1/(t(2) - t(1)));   % assumes t is periodic

key = 0;
FINISHED = 27;     % escape
PAN_LEFT = 28;     % left arrow
PAN_RIGHT = 29;    % right arrow
SCROLL_LEFT = 11;  % page down
SCROLL_RIGHT = 12; % page up
DELETE_1 = 8;      % backspace
DELETE_2 = 127;    % delete
ZOOM_IN = 30;      % up arrow
ZOOM_OUT = 31;     % down arrow
FAST_ZOOM_IN = 46;  % decimal
FAST_ZOOM_OUT = 48;  % zero
COMPLETE_PLOT_1 = 97; % 'a' key
COMPLETE_PLOT_2 = 65; % 'A' key
% note: 1, 2, and 3 are reserved for mouse button(s)

num_pick = 0;    % number of points picked
num_regions = 0; % number of regions picked
num_highs = 0;   % number of regions highlighted
os = 1;          % offset of start of current trace, in samples
w = l - 1;       % width of trace, in samples
redraw = 1;      % flag for plotting
mf = 1;          % magnification factor

while (key ~= FINISHED)

   if (redraw == 1)
```

```
    df = floor(w/2000);
    if (df < 1)
       df = 1;
    end
    tr = t(os:df:os+w);
    pr = pos(os:df:os+w);

% leave some blank space above and below trace for aesthetics
    mx = max(pr);
    if (mx < 0)
       mx = mx * 0.9;
    else
       mx = mx * 1.1;
    end
    mn = min(pr);
    if (mn < 0)
       mn = mn * 1.1;
    else
       mn = mn * 0.9;
    end

    hold off
    axis([tr(1) tr(length(tr)) mn mx]);
     if (colour == 'y')

       % plot eye position signal in black, solid
        plot(tr,pr,'w')

       % plot picked regions in blue, dash-dotted
       hold on
       for i=1:num_regions
          t3 = (regions(i,1) - 1)/sample;
            plot([t3,t3],[mn,mx],'b-.')
          t4 = (regions(i,2) - 1)/sample;
            plot([t4,t4],[mn,mx],'b-.')
            plot([t3,t4],[mn,mx],'b-.')
       end
       % plot currently picked point in blue, dotted
       if (num_pick == 1)
           plot([t1,t1],[mn,mx],'b:')
       end

       % plot highlighted regions in green, solid
       for i=1:num_highs
          t3 = (highs(i,1) - 1)/sample;
            plot([t3,t3],[mn,mx],'g')
          t4 = (highs(i,2) - 1)/sample;
            plot([t4,t4],[mn,mx],'g')
            plot([t3,t4],[mn,mx],'g')
       end

    else

       % plot eye position signal in solid
```

```
      plot(tr,pr,'-')

      % plot picked regions in dash-dotted
      hold on
      for i=1:num_regions
         t3 = (regions(i,1) - 1)/sample;
          plot([t3,t3],[mn,mx],'-.')
         t4 = (regions(i,2) - 1)/sample;
          plot([t4,t4],[mn,mx],'-.')
          plot([t3,t4],[mn,mx],'-.')
      end
      % plot currently picked point in dotted
      if (num_pick == 1)
          plot([t1,t1],[mn,mx],':')
      end

      % plot highlighted regions in dashed
      for i=1:num_highs
         t3 = (highs(i,1) - 1)/sample;
          plot([t3,t3],[mn,mx],'--')
         t4 = (highs(i,2) - 1)/sample;
          plot([t4,t4],[mn,mx],'--')
          plot([t3,t4],[mn,mx],'--')
      end
   end

    text(.7,.93,['magnification = ',int2str(round(mf)),' X'],'sc')
   redraw = 0;

end

[x,y,key] = ginput(1);

if (key==ZOOM_IN)     % increase magnification factor

   old=mf;
   mf=min(old*2,max(old,floor(l/100)));
   if mf==old    % maximum magnification of 100X
      redraw=0;
   else
      redraw=1;
      w=floor(l/mf);
   end

elseif (key == FAST_ZOOM_IN)   % fast two-point zoom

   % first point of region to zoom into
   [t3,y,key] = ginput(1);
   if ((key ~= DELETE_1) & (key ~= DELETE_2))

      % bounds check on first point of region
      if (t3 < tr(1))
         t3 = tr(1);
      elseif (t3 > tr(length(tr)))
```

```
      t3 = tr(length(tr));
    end
    x3 = 1 + round(t3 * sample);
    t3 = (x3 - 1)/sample;

    % display first point
    hold on
     if (colour == 'y')
        plot([t3,t3],[mn,mx],'r:');
    else
        plot([t3,t3],[mn,mx],':');
    end
    hold off
    redraw = 1;

    % second point of region to zoom into
    [t4,y,key] = ginput(1);

    % allow user to abort zoom via delete key
    if ((key ~= DELETE_1) & (key ~= DELETE_2))

      % bounds check on second point of region
      if (t4 < tr(1))
        t4 = tr(1);
      elseif (t4 > tr(length(tr)))
        t4 = tr(length(tr));
      end
      x4 = 1 + round(t4 * sample);
      t4 = (x4 - 1)/sample;

      % display second point
      hold on
       if (colour == 'y')
          plot([t4,t4],[mn,mx],'r:');
      else
          plot([t4,t4],[mn,mx],':');
      end
      hold off

      % swap order of points if needed
      if (x4 < x3)
        old = x4;
        x4 = x3;
        x3 = old;
      end

      % calculate new magnification parameters
      if (x3 ~= x4)
        os = x3;
        w = x4 - x3;
        mf = l/w;
      end
    end
end
```

```
elseif (key==ZOOM_OUT)  % decrease magnification

   if (mf == 1)  % already completely zoomed out
      redraw = 0;
   else
      redraw=1;
      old=mf;
      mf=max(floor(old/2),1);
      w=floor(l/mf);
      if (w >= l)
         w = l - 1;
      end
      if ((os+w)>l)
         os=floor(max(1,l-w));
      end
   end

elseif ((key == COMPLETE_PLOT_1) | (key == COMPLETE_PLOT_2) | (key ==
FAST_ZOOM_OUT))  % display entire plot

   os = 1;
   mf = 1;
   w = l - 1;
   redraw = 1;

elseif (key==PAN_RIGHT)  % increase offset by quarter-screen

   old=os;
   os=floor(max(1,min(l-w,os+0.25*w)));
   if old==os      % already panned to end
      redraw=0;
   else
      redraw=1;
   end

elseif (key==PAN_LEFT)  % decrease offset by quarter-screen

   old=os;
   os=floor(max(1,os-0.25*w));
   if os==old      % already panned to beginning
      redraw=0;
   else
      redraw=1;
   end

elseif (key==SCROLL_RIGHT)  % jump display one screenful right

   old=os;
   os=floor(max(1,min(os+w,l-w)));
   if os==old      % already panned to end
      redraw=0;
   else
      redraw=1;
```

```
   end

elseif (key==SCROLL_LEFT)  % jump display one screenful left

   old=os;
   os=floor(max(1,os-w));
   if old==os      % already panned to beginning
      redraw=0;
   else
      redraw=1;
   end

elseif ((key==DELETE_1) | (key==DELETE_2))

   if (num_pick > 0)  % wipe out currently picked point
      num_pick = 0;
      redraw = 1;
   elseif (num_highs > 0)  % wipe out highlit regions
      for i=1:num_highs
         index = InList(highs(i,1),regions);
         regions = DeleteRow(index,regions);
      end
      num_regions = num_regions - num_highs;
      num_highs = 0;
      clear highs
      redraw = 1;
   end

elseif (key==1) | (key==2) | (key==3)   % up to three-button mouse input

   if (num_pick == 0)  % this is the first picked point

      % bounds check on picked point
      if (x < tr(1))
         x = tr(1);
      elseif (x > tr(length(tr)))
         x = tr(length(tr));
      end

      % convert time value to sample number
      x1 = 1 + round(x * sample);

      % see if point is in a selected region
      index1 = InList(x1,regions);
      if (index1 > 0)
         index2 = InList(x1,highs);
         if (index2 > 0)  % de-highlight region
            num_highs = num_highs - 1;
            highs = DeleteRow(index2,highs);
            redraw = 1;
         else  % highlight region for future deletion
            x1 = regions(index1,1);
            x2 = regions(index1,2);
            t1 = (x1 - 1)/sample;
```

```
        t2 = (x2 - 1)/sample;
        num_highs = num_highs + 1;
        highs(num_highs,:) = [x1 x2];
        if (colour == 'y')
          hold on
           plot([t1,t1],[mn,mx],'g');
           plot([t2,t2],[mn,mx],'g');
           plot([t1,t2],[mn,mx],'g');
        else
          hold on
           plot([t1,t1],[mn,mx],'-');
           plot([t2,t2],[mn,mx],'-');
           plot([t1,t2],[mn,mx],'-');
        end
      end
    else  % point is not already in a selected region
      % display as first point of region being selected
      t1 = (x1 - 1)/sample;
      num_pick = 1;
      hold on
      if (colour == 'y')
          plot([t1,t1],[mn,mx],'b:');
      else
          plot([t1,t1],[mn,mx],':');
      end
      hold off
    end
  elseif (num_pick == 1)  % second picked point

    % bounds check on picked point
    if (x < tr(1))
      x = tr(1);
    elseif (x > tr(length(tr)))
      x = tr(length(tr));
    end

    % convert time value to sample number
    x2 = 1 + round(x * sample);
    t2 = (x2 - 1)/sample;

    if (x2 == x1)  % cannot have interval of zero width
      num_pick = 0;
      redraw = 1;
    else
      hold on
      if (colour == 'y')
          plot([t2,t2],[mn,mx],'b:');
      else
          plot([t2,t2],[mn,mx],':');
      end
      if (x2 < x1)   % order picked points
        old = x1;
        x1 = x2;
        x2 = old;
```

```
            old = t1;
            t1 = t2;
            t2 = old;
          end
          num_pick = 0;
          if (colour == 'y')
             plot([t1,t2],[mn,mx],'b:');
          else
             plot([t2,t2],[mn,mx],':');
          end

          % add picked region to list
          num_regions = num_regions + 1;
          regions(num_regions,:) = [x1 x2];
          hold off
        end
      end
    end
end

clear i index index1 index2 t1 t2 x1 x2 x y mn mx mnv sample
clear highs num_highs num_regions old num_pick
clear key redraw os w mf l df pr tr
return;
```

```
%prep_data
%
% Written by D. Balkwill 9/26/90
%
% This prepares data for processing, without using the 'batch'
% script.  Run parameters are input, data is rescaled from
% arbitrary units to degrees (using previous calibration), and
% data is moved to the appropriate directories.

nysa_path = get_path;

run_code = input('Enter Run Code: ','s');
eval(['load ',nysa_path,'bookkeeping:vel_filter.mat']);
eval(['save ',nysa_path,'bookkeeping:run_code run_code']);
eval(['load ',nysa_path,'bookkeeping:dim']);

offset1 = 0;
scale1 = 1;
offset2 = 0;
scale2 = 1;
offset3 = 0;
scale3 = 1;

if (exist([nysa_path,'bookkeeping:cal']) == 2)
   eval(['load ',nysa_path,'bookkeeping:cal']);
end

if (dim >= 1)
   ans = input(['Enter Axis#1 Calibration Factor in deg/unit [default =
',sprintf('%7.4f',scale1),']    ']);
   if (isempty(ans) == 0)
      scale1 = ans;
      offset1 = 0;
   end
end

if (dim >= 2)
   ans = input(['Enter Axis#2 Calibration Factor in deg/unit [default =
',sprintf('%7.4f',scale2),']    ']);
   if (isempty(ans) == 0)
      scale2 = ans;
      offset2 = 0;
   end
end

if (dim >= 3)
   ans = input(['Enter Axis#3 Calibration Factor in deg/unit [default =
',sprintf('%7.4f',scale3),']    ']);
   if (isempty(ans) == 0)
      scale3 = ans;
      offset3 = 0;
   end
end
```

scale_data

clear run_code nysa_path dim cal A B sample
clear offset1 noise1 scale1 offset2 noise2 scale2 offset3 noise3 scale3

```
% process
%
% D.M. Merfeld and D. Balkwill 9/26/90
%
% This script uses an acceleration-based algorithm for detection,
% and interpolation across, fast phases, thus yielding an estimated
% slow phase velocity profile. If the acceleration thresholds are
% not specified, they are set on a pseudo-statistical basis.

%get NysA path if necessary
if (exist('nysa_path') ~= 1)
   nysa_path = get_path;
end

%load in the number of dimensions of data to analyze
eval(['load ',nysa_path,'bookkeeping:dim'])

%calculate velocity and acceleration for each dimension
vel_filt %Performs actual position to velocity filtering
acc_filt %Performs actual velocity to acceleration filtering

%load in acceleration thresholds
eval(['load ',nysa_path,'bookkeeping:thresh.mat'])
if ((thresh_acc < 0) | (thresh_end < 0))
   x = mag_acc;
   mn = mean(x);
   med = median(x);
   l = length(x);
   for i=1:l
      if (x(i) > (med + mn))
         x(i) = med;
      end
   end
   sd = std(x);
   thresh_acc = floor(mn + 2*sd);
   thresh_end = floor(med + sd);
   fprintf('\nAcceleration thresholds were not specified.');
   fprintf(['\nUsing peak threshold = ',int2str(thresh_acc)]);
   fprintf(['\n      end threshold = ',int2str(thresh_end)]);
   fprintf('\n');
   clear mn med sd x
end

%get list of detected events in 'flag'
heart
eval(['save ',nysa_path,'Data_Out:flag flag'])

%interpolate across events
remove

clear spv1 spv2 spv3 flag
clear sample mag_acc thresh_acc thresh_end dim
```

```
% remove
%
%       Takes all events which have been classified as fast phases and removes
%       them to yield slow phase eye velocity. The event flags from 'heart' are used
% by 'fill' to interpolate across the fast phases. The velocities are loaded
% from the disk, if necessary, and the interpolated SPV's are saved on the disk.
% D. Balkwill 9/26/90

if (exist('flag') ~= 1)
   eval(['load ',nysa_path,'bookkeeping:flag'])
end
if (exist('vel1') ~= 1)
   eval(['load ',nysa_path,'bookkeeping:vel1'])
end
x = vel1;
clear vel1
fill
spv1 = x;
eval(['save ',nysa_path,'Data_Out:spv1 spv1'])

if (dim >= 2)
   if (exist('vel2') ~= 1)
      eval(['load ',nysa_path,'bookkeeping:vel2'])
   end
   eval(['load ',nysa_path,'bookkeeping:vel2'])
         x = vel2;
   clear vel2
   fill
   spv2 = x;
   eval(['save ',nysa_path,'Data_Out:spv2 spv2'])
end

if (dim >= 3)
   if (exist('vel3') ~= 1)
      eval(['load ',nysa_path,'bookkeeping:vel3'])
   end
   eval(['load ',nysa_path,'bookkeeping:vel3'])
   x = vel3;
   clear vel3
   fill
   spv3 = x;
   eval(['save ',nysa_path,'Data_Out:spv3 spv3'])
end

clear x
```

```
%scale_data
%
% D. Balkwill  9/26/90
%
% Subtracts a DC offset from the position data, and converts
%  arbitrary units into degrees, by using the calibration factors.
% Moves the position data from the user-defined data folder
%  into the NysA Data_In folder.

% Modification: 10/28/90
% Calls optional 'condition' script to perform signal conditioning.
% Current implementation is two passes of 'OS_lin2'
%  order statistic script with window lengths (6,10).
% Original data is unaffected.

if (exist('nysa_path') ~= 1)
   nysa_path = get_path;
end

file_specs

if (exist('dim') ~= 1)
    eval(['load ',nysa_path,'bookkeeping:dim']);
end

if (dim >= 1)
   in_file = file_name(Pos1_File,run_code);
   eval(['load ',data_path,in_file]);
    eval(['pos1 = (',Pos1_Var,' - offset1) * scale1;']);
   if (exist('condition') == 2)
      pos1 = condition(pos1);
   end
    eval(['save ',nysa_path,'Data_In:pos1 pos1']);
   eval(['clear ',Pos1_Var]);
   clear pos1
end

if (dim >= 2)
   in_file = file_name(Pos2_File,run_code);
   eval(['load ',data_path,in_file]);
    eval(['pos2 = (',Pos2_Var,' - offset2) * scale2;']);
   if (exist('condition') == 2)
      pos2 = condition(pos2);
   end
    eval(['save ',nysa_path,'Data_In:pos2 pos2']);
   eval(['clear ',Pos2_Var]);
   clear pos2
end

if (dim >= 3)
   in_file = file_name(Pos3_File,run_code);
   eval(['load ',data_path,in_file]);
    eval(['pos3 = (',Pos3_Var,' - offset3) * scale3;']);
   if (exist('condition') == 2)
```

```
      pos3 = condition(pos3);
   end
    eval(['save ',nysa_path,'Data_In:pos3 pos3']);
    eval(['clear ',Pos3_Var]);
    clear pos3
end

clear_specs
clear in_file
```

```
function [scale,noise,offset] = three_point(t,pos,colour)

% This script inputs the angular deviations from the user, and
% calls the 'pick_regions' script so that the fixation regions
% can be selected. For each deviation, the average value over
% all of the regions is calculated. The calibration factor for
% each axis is calculated as the angular difference between the
% positive and negative deviations (in degrees), divided by the
% difference between mean positive and negative deviations (in
% arbitrary units). The calculated mean values are displayed
% to the user graphically as dotted lines, for inspection.
%
% If a zero fixation point is specified, it is used to
% determine the offset value.
%
% The noise is estimated by taking the root-mean-square value of
% the fluctuations about all selected regions.
%
% D. Balkwill 11/27/90

pos_deg = input('Specify eye movement in degrees, positive trace deflection: ');
if isempty(pos_deg)
  disp('  Default calibration target assumed to be 10 degrees.');
  pos_deg = 10;
end
neg_deg = input('Eye displacement for negative trace deflection: ');
if isempty(neg_deg)
  disp('  Default calibration assumed -10 degrees.');
  neg_deg = -10;
end
if (pos_deg == neg_deg)
  disp('Calibration range cannot be zero, Symmetrical calibration assumed.');
  neg_deg = -1 * pos_deg;
end

% select positive trace deflection regions
disp('');
disp('Use mouse to select flat-top regions of positive trace deflection.');
pos_regions = pick_regions(t,pos,colour);
[m,n] = size(pos_regions);
if (m == 0)
  error('No positive trace deflection flat-top regions, Cannot calibrate.');
end
for i=1:m
   xpos = [xpos ; pos(pos_regions(i,1):pos_regions(i,2))];
end
mpos = mean(xpos);
hold on
if (colour == 'y')
   plot([t(1),t(length(t))],[mpos,mpos],'r:');
else
   plot([t(1),t(length(t))],[mpos,mpos],':');
end
hold off
```

```
% select negative trace deflection regions
disp('Now select flat-top regions of negative trace deflection.');
neg_regions = pick_regions(t,pos,colour);
[m,n] = size(neg_regions);
if (m == 0)
  error('Cannot calibrate, no negative trace deflections selected.');
end
for i=1:m
   xneg = [xneg ; pos(neg_regions(i,1):neg_regions(i,2))];
end
mneg = mean(xneg);
hold on
if (colour == 'y')
   plot([t(1),t(length(t))],[mneg,mneg],'r:');
else
   plot([t(1),t(length(t))],[mneg,mneg],':');
end
hold off

% calculate scale factor in deg/unit
scale = (pos_deg - neg_deg) ./ (mpos - mneg);

% select optional zero trace deflection regions
y=input('Is there a zero degree calibration target? (y,n) [default = n]  ','s');
if isempty(y)
   y='n';
end
xzero = [];
if y=='y' | y=='Y'
   disp('Select regions in which subject fixated on zero reference.');
   zero_regions = pick_regions(t,pos,colour);
   [m,n] = size(zero_regions);
   if (m == 0)
      offset = 0;
   else
      for i=1:m
         xzero = [xzero ; pos(zero_regions(i,1):zero_regions(i,2))];
      end
      offset = mean(xzero);
      hold on
       if (colour == 'y')
          plot([t(1),t(length(t))],[offset,offset],'r:');
      else
          plot([t(1),t(length(t))],[offset,offset],':');
      end
      hold off
   end
else
   offset = 0;
end

% display positive and negative mean values
hold on
```

```
if (colour == 'y')
   plot([t(1),t(length(t))],[mneg,mneg],'r:');
   plot([t(1),t(length(t))],[mpos,mpos],'r:');
else
   plot([t(1),t(length(t))],[mneg,mneg],':');
   plot([t(1),t(length(t))],[mpos,mpos],':');
end
hold off

% estimate noise, in rms degrees
xpos = xpos - mean(xpos);
xneg = xneg - mean(xneg);
if (isempty(xzero) == 0)
   xzero = xzero - mean(xzero);
end
x = [xpos ; xneg ; xzero ];
noise = scale * sqrt(mean(x.*x))

clear pos_regions neg_regions zero_regions xpos xneg xzero x
clear i m n y neg_deg pos_deg
return;
```

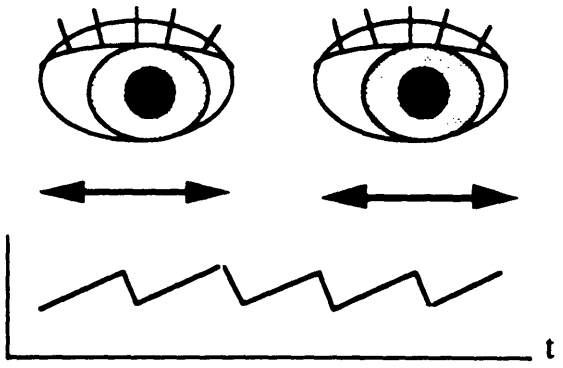
```
%VEL_FILT This program differentiates eye angular velocity to yield eye
%       angular acceleration.  This is done for each of three DOF.

%       D.M. Merfeld 4/17/89
% D. Balkwill 8/27/90

eval(['load ',nysa_path,'bookkeeping:vel_filter.mat'])

eval(['load ',nysa_path,'Data_in:pos1'])
x = pos1;
clear pos1
filtzero
vel1 = x;
eval(['save ',nysa_path,'bookkeeping:vel1 vel1'])

if (dim >= 2)
    eval(['load ',nysa_path,'Data_In:pos2'])
          x = pos2;
   clear pos2
          filtzero
          vel2 = x;
    eval(['save ',nysa_path,'bookkeeping:vel2 vel2'])
end

if (dim >= 3)
           eval(['load ',nysa_path,'Data_In:pos3'])
         x = pos3;
   clear pos3
         filtzero
           vel3 = x;
    eval(['save ',nysa_path,'bookkeeping:vel3 vel3'])
end

clear A B x
```

Appendix E

NysA User's Manual

NysA V2.0

Nystgamus Analysis System



USER'S MANUAL

Documentation by

M. David Balkwill
J. Ted Liefeld

Massachusetts Institute of Technology
Man - Vehicle Laboratory

NysA User's Manual V2.0

1. Introduction

NysA is a Nystagmus Analysis system developed at MIT for vestibular and oculomotor research using the Apple Macintosh. The user provides original data files consisting of sampled horizontal, vertical, and/or torsional eye position time series. The principal goal of NysA is to then determine an estimate of the slow phase eye velocity, by differentiating the eye position and interpolating across fast phases. Other data, such as eye velocity and acceleration, are also calculated. Two algorithms are available; an improved version of the Merfeld algorithm, and an order statistic algorithm. These will be discussed in Section 14 and 15 respectively.

NysA is functionally a set of MatLab scripts. MatLab is a software package for scientific and engineering numeric computation; a licensed copy must be obtained from MathWorks, Inc. before NysA can be used. By piggy-backing NysA on MatLab, the user can easily modify or expand NysA as desired, customizing it to an individual experiment. Since MatLab is available on a number of different platforms (including DOS and unix systems), NysA is portable.

No assumptions are made about the nature of the experimental stimulus, the method of recording eye position, or the number of axes of eye position. Also, a variety of sampling frequencies are permissible. As such, NysA is useful for many different experiments.

Sections 2 and 3 describe the equipment and steps to install NysA. Section 4 explains the fast phase detection and removal algorithm. Sections 5 and 6 outline the structure of the folders and the file names which NysA uses. Sections 7 through 13 describe the principal scripts which comprise NysA; each of these scripts can be executed to perform a certain step in the analysis. Appendix A steps through the installation procedure and the *nysa_demo* script to describe the main features of NysA in the context of a typical analysis procedure. Appendix B outlines the various sets of sample data which are included on the distribution disk.

2. System Requirements

Since NysA is a set of MatLab scripts, it can be used on any Macintosh system which is equipped with MatLab. In particular, it has been run on the Mac SE, II, IIci, and IIfx, with System 6.0.5 and MultiFinder. A large capacity SCSI hard disk will be required, along with a cartridge disk or tape unit for backing up the data. The NysA scripts can also be run on a Mac SE equipped with Edu-Matlab but this is not recommended due to slow execution and file size limitations.

As the order statistics based fast phase removal algorithm (*AATM*) has been implemented as a MatLab mex file, MatLab version 3.5+ is required for this data processing to be available.

A color monitor is not required, but is recommended, due to the use of interactive graphics in NysA; if a monochrome monitor is used, different line types can be used in lieu of different colors.

The amount of memory which is required will be dictated by the size of the data files which are to be analyzed. For most realistic applications, a minimum of four megabytes of memory will probably be needed. A RAM cache can be used to decrease the time for

loading and saving data.

It is recommended that the keyboard repeat delay is set to medium or long for use with NysA.

It is also recommended that the user become familiar with the use of MatLab, and the MatLab programming enviorment prior to using the NysA system.

To date, no conflicts have been found between NysA, MatLab, and other applications or system INITS and CDEVS.

Note that NysA is not a data collection program. The original eye position data must be collected with another program.

3. Installation

1. First, ensure that MatLab is already installed on the hard disk on which NysA is to be installed. About 750 K of disk space will be required for NysA and the included sample data.

2. Insert the distribution diskette into the floppy drive and copy the entire distribution diskette onto the disk by dragging the disk icon. (There is insufficient space on a single diskette to run the NysA package). The name of this folder should then be *NysA_V2.0.* .

3. Move the *NysA_V2.0* folder to the same level as the MatLab application.

4. Launch the MatLab application.

5. Use the *Set Path* menu command to add the *NysA_V2.0* and *NysA_V2.0:scripts* folders to the MatLab search path. The "Auto Set" option will not add the *scripts* folder; it must be added manually.

6. Run the *init* script once to set up the NysA path. This path should be the same as the first line which was added to the MatLab search path (pointing to the *NysA_V2.0* folder, not to the *scripts* folder). The full meaning of this path will be described in Section 5. The *init* script will request values for other parameters as well; these should be left at the default values for now. These will be described in Section 7.

7. Open the *file specs* script to set up the NysA data path. This path should lead to the folder holding the data files that you have just copied from the distribution diskette.

The NysA package is now ready to run. For a quick tour of the main features, turn to Appendix A and run the enclosed *nysa_demo* script.

Advanced MatLab users will find that the placement and name of the NysA folder are somewhat arbitrary. However, this installation procedure is straightforward, does not require an advanced knowledge of MatLab, and eliminates potential ambiguities associated with the MatLab search path.

4. NysA Folder Usage

The distribution diskette should contain six folders and a readme file (this manual). The

folders are *bookkeeping, Data_In, Data_Out, filters, Sample_Data,* and *scripts*. These names should not be changed. When NysA is installed on the system, all of these folders should be within a single folder called *NysA_V2.0*. This folder is the "root folder"; the full specification of this folder, including all higher level folder names and the hard disk name, is the NysA path which was mentioned in Section 3. For instance, if the name of the hard disk is *disk* and the MatLab folder is at the top level, then the NysA path would be *disk:MatLab:NysA_V2.0*. This path name may optionally end in a semicolon. Once this path is specified in *init*, it is saved on the hard disk as *nysa_path*, and will be the default from then on; it does not need to be re-entered every time that NysA is run.

The *bookkeeping* folder used by NysA to save parameters such as the sampling frequency and the acceleration thresholds. The *Data_In* and *Data_Out* folders are used for local storage of the eye position (input data) and slow phase velocity (output data). The original data should reside in a separate folder (see Section 6) where it cannot be contaminated or overwritten. The *filters* folder contains the digital filter coefficients for the velocity and acceleration differentiation filters; the adventurous user may customize the filters by replacing the standard coefficients. The *Sample_Data* folder contains some sample data files for introduction to the NysA package (see Appendix A); it can be safely deleted when it is no longer needed. The *scripts* folder contains the various MatLab scripts which constitute NysA; it is worthwhile to glance at the names of these scripts so that duplicate file names are not used elsewhere.

5.Execution

The execution of the NysA scripts is as follows;

1. Check *file_specs* to assure proper data path and variable names.

2. *init*

3. *calibrate*

4. *prep_data*

5. either *process* or *AATM*

6. *backup_data* (if desired)

7. *edit_spv*

Each of these scripts will be discussed in detail below.

6. File and Variable Name Specification (*file_specs*)

The *file_specs* script allows the user to have control over the names of the data file and variable names, and the location where they are stored. This file is meant to be customized by the user. It contains a number of assignment statements of the form

variable = 'name';

Only the names within single quotation marks are to be modified. The variable names on the left side of the assignments are internal names which are used by NysA and must not be changed.

It is very important to check the *file_specs* script at the beginning of each new NysA session to assure the correct file path and variable names are present. A common user error is the running of NysA with incorrect variable name or file path names.

The *data_path* is analogous to the *nysa_path* in format; it is the complete specification of the folder in which the original eye position data is stored. Following the example of Section 5 above, the NysA demo should be run with the data path set to *disk:MatLab:NysA_V2.0:Sample_Data*. The eye position data is copied from this folder to the *Data_In* folder before NysA analyzes it. Similarly, the velocity and acceleration data is copied from the *Data_Out* folder to the data path.

The input (eye position) data files are assumed to be MatLab compatible and stored with a variable name. If more than one axis of input data is recorded, it is assumed that different channels are stored in separate files with separate variable names. For the input data, file and variable names should be, but need not be, distinct; however, the output file and variable names must be. File names must be specified for eye position (input data), eye velocity (output data), and slow phase eye velocity (output) for each axis of data; names do not need to be chosen for axes for which there is no data. Only one file name must be specified for eye acceleration (output) since the magnitude of the acceleration vector is calculated (see Section 4). Names must also be specified for calibration files (see Section 8); although they also contain eye position time series (input data), the user may wish to name them differently from stimulus runs.

A "run" is considered to be a single experimental trial, from which a single set of eye position time series data results, one file for each axis. The file names for each run are assumed to be coded by a unique identifier, henceforth referred to as a "run code". The remainder of the file names are used to distinguish between different types of data. For example, within the *Sample_Data* folder, there are two files called *N45.eogh* and *N45.eogv*. They contain horizontal and vertical EOG data respectively for a single run (in this case a calibration). *N45* is the run code and the *.eogh* and *.eogv* extensions are used to distinguish between different axes of information. *N46* is another run code for which data is included.

The run code is denoted in the file names as a '#' symbol. NysA will prompt the user for a run code when data is to be analyzed. Whatever the user enters will be substituted for the '#' symbol to create the appropriate file names.

Corresponding variable names must be specified for each of the aforementioned file names. The variable names can include a run code if desired, although they do not in this example. See the description of the demo in Appendix A if this is unclear.

The variable and file names which the user selects must not be the same as the internal variable names used by NysA; otherwise, results are unpredictable.

7. Initialization (*init*)

Several parameters will be global to a large set of data. The user initializes them by running the *init* script. This script should be run whenever one of these parameters is to be changed. If more than one person is using NysA on the same machine, and their data have different parameters, it is a good idea to run *init* at the beginning of each NysA session to ensure that the proper parameters have been set. These parameters are saved on the disk for reference, and are used as the future defaults.

The first parameter is the NysA path, which has been described fully in Section 5. It has no default value, and must be specified the first time that NysA is run or whenever the path is changed.

The second parameter is the number of axes for which eye position data was recorded (e.g. horizontal and vertical EOG would be 2 axes). The default for this value is one, unless it has been set previously.

The third parameter is the pair of acceleration thresholds, which were described in Section 4. The default value is negative, which is displayed as *<unknown>*, unless previously specified.

The fourth parameter is the sampling frequency at which the eye position data was recorded. The only allowable frequencies at present are 60, 100, 120, 128, and 200 Hz. The default is 128 Hz, unless previously specified.

The final parameter which must be specified is whether or not a color monitor is available. The graphics sections of NysA make use of different colors for different traces, or alternatively use different line types if only a monochrome monitor is present. The default is to assume a color monitor, unless previously specified.

8. Eye Position Calibration (*calibrate*)

Overview

An eye position time series, as sampled by an analog-to-digital (A/D) board, is often obtained in arbitrary units which correspond to a number of sample counts. For example, a twelve-bit A/D board may return integer values between -2048 and +2047. Before any meaningful analysis can be made on the data, it must be converted to physical units of degrees by determining a calibration factor. A common calibration procedure is to have the subject gaze at a number of targets whose positions are known and to determine the resulting voltage values, thereby obtaining a calibration factor.

The *calibrate* script assumes that a three-point or a two-point calibration is performed for each axis of data. Thus, for each axis, there will be a time series containing three flat plateau regions (three-point), corresponding to the times when the gaze was directed at the zero point, the positive deflection point and the negative deflection point. For two-point calibrations, the zero point calibration target is usually omitted. The angular deviations for the fixation points (both positive and negative deflections from the zero point) must be entered by the user; these values will usually be the same, but need not be. The user then selects the regions of the calibration at which the gaze was directed to these points. This yields values for two fixation points in degrees and in the original units. The calibration factor is then calculated as the ratio of the difference in degrees to the difference in original units.

If the zero point is one of the fixation points (i.e. the center of the target), it can be selected in the same way as the other points. It is used to determine an offset in the data so that actual eye position can be determined; if it is not specified, offset will be calculated as the mean of the positive and negative deviation values. The zero point is not used in the calculation of the calibration factor.

The RMS noise of the picked region around the calculated calibration factor is calculated to

provide an estimate of the 'noisiness' in the data.

Zoom and Pan

The time series for a calibration may be much longer than can be easily displayed. As such, a number of zoom and pan operations have been implemented. They apply only to the horizontal axis (time); vertical scaling is automatic.

The up and down arrow keys control the "magnification factor", which is displayed at the top right of the graphics window. The up arrow zooms in, increasing the magnification by a factor of two. The displayed graph is anchored at the left side; that is, the left half of the trace on the screen is redisplayed. The down arrow zooms out, decreasing the magnification by a factor of two; again, the graph is anchored at the left. There is a limit to how far the graph can be magnified; specifically, at least 100 points of data must be displayed.

The page down and page up keys scroll the graph by a full screen width, left and right respectively. The left and right arrow keys pan the graph by one-quarter of a screen width, left and right respectively.

A new feature is the fast zoom. First, press the decimal key and then select two points with the mouse. The graph is zoomed into the region defined by the chosen points. The magnification factor is updated accordingly. The zero key is used to zoom out completely, resetting the magnification to 1.

Picking Fixation Points

The flat plateau regions are selected interactively by the user. When the *calibrate* script is run, it will prompt for a run code which specifies which data files are to be loaded (see Section 6). Each axis is calibrated individually and sequentially.

The positive and negative angular deflections for that axis are entered by the user, the time series is displayed in black (solid line), and the user is requested to select the regions corresponding to a positive deflection. A region is selected by using the mouse to pick two points on the graph. Only the time coordinate of each point is of importance here. The region is marked on the graph by two vertical lines and a diagonal in blue (dotted). If a number of regions of the time series correspond to the positive fixation point, they can all be selected and each will be marked as it is selected.

If a marked region was picked improperly, it can be de-selected. Picking a point within that region will cause the marking to be highlighted in green (solid). Picking a second time within that region will remove the highlighting (i.e. highlighting is implemented as a toggle switch). A number of regions can be highlighted at once. The delete key is then used to de-select all highlighted regions. All markings for the de-selected regions are removed. Note that a region is not de-selected until the delete key is pressed.

The escape key is used to signify that all desired regions have been picked. The value (in original units) for a fixation point is calculated as the arithmetic mean of the values of all points in all selected regions.

The negative fixation point is then selected similarly, as is (optionally) the zero point.

9. Data Preparation (*prep_data*)

The *prep_data* script prompts the user for a run code, and for a calibration factor for each axis. The original eye position data for that run is then loaded from the *data_path* and converted to degrees. If the original eye position data is already in degrees, the calibration factor should be simply specified as 1. The converted data is then saved in the NysA *Data_In* folder. The run code is saved in the *bookkeeping* folder.

A new NysA feature performs signal conditioning with the *condition* script. It is currently set up to apply non-linear order-statistic filtering to the eye position time series, thereby reducing noise in the signal and sharpening the peaks of nystagmus. A detailed discussion of order-statistic filtering is beyond the scope of this section, but it is implemented with the algorithm used by Engelken and Stevens, 1990.

The user can customize the *condition* script to perform whatever conditioning is desired, taking advantage of MatLab's signal processing toolbox. This script is applied to each axis of data before it is saved in the *Data_In* folder. The original eye position data files are not modified in any way at any point in the NysA system.

10. Data Processing (*process, AATM*)

The *process* script implements the acceleration based Merfeld algorithm which is described in Section 14. The input eye position time series are loaded from the *Data_In* folder, and the output data are saved in the *Data_Out* folder. This script does not require any user interaction.

The *AATM* script implements the order statistic filter algorithm which is described in Section 15. It reads the *data_path* from the *bookkeeping* folder, and asks the user to enter the sample rate and the run_code. The filtered eye position, velocity, and slow phase velocity are then saved in the same folder as the original data.

11. Backing Up (*backup_data*)

The *backup_data* script copies the eye velocity, slow phase eye velocity, and eye acceleration time series into the original *data_path*. It loads the run code from the *bookkeeping* folder to determine the names of the output files. If files with those names already exist in the *data_path*, they are overwritten.

12. Batch Processing (*batch*)

Once the calibration factors are known for a number of runs, the next step in processing the data (running the automated algorithm) is purely computational with no need for an interactive user. As such, the *batch* script allows the user to enter an arbitrary number of run codes and corresponding calibration factors at once; the list is terminated when a null run code is entered. Then, the *prep_data, process,* and *backup_data* scripts are automatically run for each run code.

13. Manual Editing (*edit_spv*)

The *edit_spv* script allows the user to interactively review the results of the automated algorithm, and to manually edit desired sections. In some cases, a fast phase may be

missed because the eye acceleration remains below the peak threshold. In other cases, the beginning and end of a fast phase may be incorrectly detected. Finally, a first order interpolation may be more appropriate than a zero order one in areas of rapidly changing slow phase velocity. The first case requires the user to specify the fast phase; the latter two cases require the user to override the automated algorithm by removing its interpolation and substituting the correct one. In any case, the user's ability to select the beginning and end of the fast phases is superior to that of the automated algorithm; as such, all interpolations in the manual editing process are first order.

The run code, and the number of the axis of data to be edited, are input by the user at the beginning of the manual editing script. This script works on only one axis of data for one run; therefore, any changes made to one axis will not be applied to other axes. In order to give the user sufficient information to make proper judgments, the eye position, eye velocity, and slow phase eye velocity are loaded from the disk and displayed. The user is required to specify whether or not the run to be edited was the last one processed by the *process* script. If this is the case, the data are loaded from the NysA folders; this will display the filtered and calibrated eye position. If this is not the last run processed, the data are loaded from the *data_path*, and an approximate calibration factor is calculated by analyzing the velocity and position data; this takes a little bit of additional time.

Once these data are loaded, the velocity and slow phase velocity are compared to determine those regions which have been classified as fast phases. This is also somewhat time-consuming, but is necessary if the automated interpolations are to be properly modified. The entire slow phase eye velocity is then plotted in black (or simply solid on a monochrome screen) and the editing process begins. If the length of the displayed data is less than ten seconds, the eye velocity is plotted in red (or dotted) and the eye position is plotted in green (or dashed); otherwise, these are not plotted because the display would be too cluttered. The vertical axis of the graph is in units of both degrees and degrees per second.

The editing process now begins. The user interface is similar to that of the calibration procedure (see Section 8). The same zoom and pan features are present, regions are selected in the same way, and the escape key is used to signify the end of the editing process.

Missed fast phases are to be specified as regions. The points to be picked by the mouse are the last "good" point before the fast phase and the first "good" point after the fast phase (i.e. points which are not part of the fast phase). The two points can be picked in any order. This corresponds to the way in which the flat plateau regions were selected in the calibration procedure. Several such regions can be specified at once.

When a region is specified in this way, it is marked and a blue (or dash-dot) line will be displayed which corresponds to the interpolation which would be made across that fast phase. This interpolation has not yet been implemented; it is merely proposed. If the proposed interpolation is acceptable, the carriage return will cause it to be implemented as the new slow phase velocity. If the proposed interpolation is not acceptable (e.g. it was off by a point or two), the delete key will reject it, the marking will be removed, and the slow phase velocity will be unchanged. If several regions are specified at once, the carriage return will cause all proposed interpolations to be implemented; however, the delete key will only reject the previous proposal.

Incorrect interpolations are deleted in the same way as regions are de-selected in the calibration procedure. Picking a point within an interpolated region will cause that region to be highlighted. Several regions can be highlighted at once. The delete key will then

cause all highlighted regions to be "deleted", by replacing the interpolated slow phase velocity with the original eye velocity. Functionally, these regions now appear as missed fast phases, and the correct interpolations are specified by the method described above.

Since the <DEL> key has more than one use, its order of priority should be clarified. If the key is pressed when a region is being selected, but only one point has been picked, then that point is deleted from consideration; that is, that region is not to be specified. The second priority is to delete any interpolations which have been selected for deletion. Finally, if no regions have been half-specified, and no interpolations have been selected for deletion, then the delete key will reject the last specified region's interpolation.

When the user has finished editing the data, this is signified by pressing the <ESC> key. The user must then specify whether or not this edited slow phase velocity is to be saved. For time considerations, the user has the ability to abort a bad editing process without having to save the data, possibly overwriting a prior and better data set. The remainder of the *edit_spv* script gives an example of how the data and parameters can be used to produce a good-looking plot of slow phase velocity versus time.

14. Acceleration Based Fast Phase Removal Algorithm

This algorithm is based on detecting peaks in the magnitude of the eye acceleration. The algorithm accepts up to three axes of eye position data (horizontal, vertical, and torsional). For example, a typical experimental setup using EOG measurement would record two axes (horizontal and vertical).

Each eye position time series is differentiated to yield the corresponding eye velocity along that axis. The eye velocity time series are then differentiated, and the vector norm is taken to yield a time series containing the magnitude of the eye acceleration. By analyzing the overall eye acceleration, information from all channels can be used at once; this aids such tasks as blink detection.

Each of these differentiation processes is implemented as a Remez equal ripple digital filter consisting of a combination of a first order derivative and a low pass filter. The number of points, and the cutoff frequency, of the filter depend upon the sampling frequency with which the original data was recorded.

Plots of typical nystagmus are shown in Figure 14.1. These are horizontal and vertical eye position as recorded by EOG, and their calculated velocities. In Figure 4.2, the acceleration magnitude, horizontal slow phase velocity (solid line), and horizontal velocity (dotted line) are shown. Notice the two blinks on the vertical eye position plot, and their effects on the velocity and acceleration.

The algorithm requires specification of two acceleration thresholds: an onset threshold and an end threshold. The acceleration time series is scanned until its magnitude exceeds the peak threshold; this signifies that a fast phase has been detected. The acceleration time series is then scanned forward and backward until the magnitude of three consecutive points is below the end threshold. Thus, two points in the time series are determined which are the estimates of the beginning and end of the fast phase. For each axis of eye velocity data, the median velocity of the five velocity points immediately prior to the fast phase is determined (to reduce the effects of noise), and this value is interpolated across the fast phase as the estimated slow phase velocity. This interpolation is a zero order hold. The eye velocity and estimated slow phase eye velocity are saved for each axis in files on the disk, as is the magnitude of the eye acceleration.

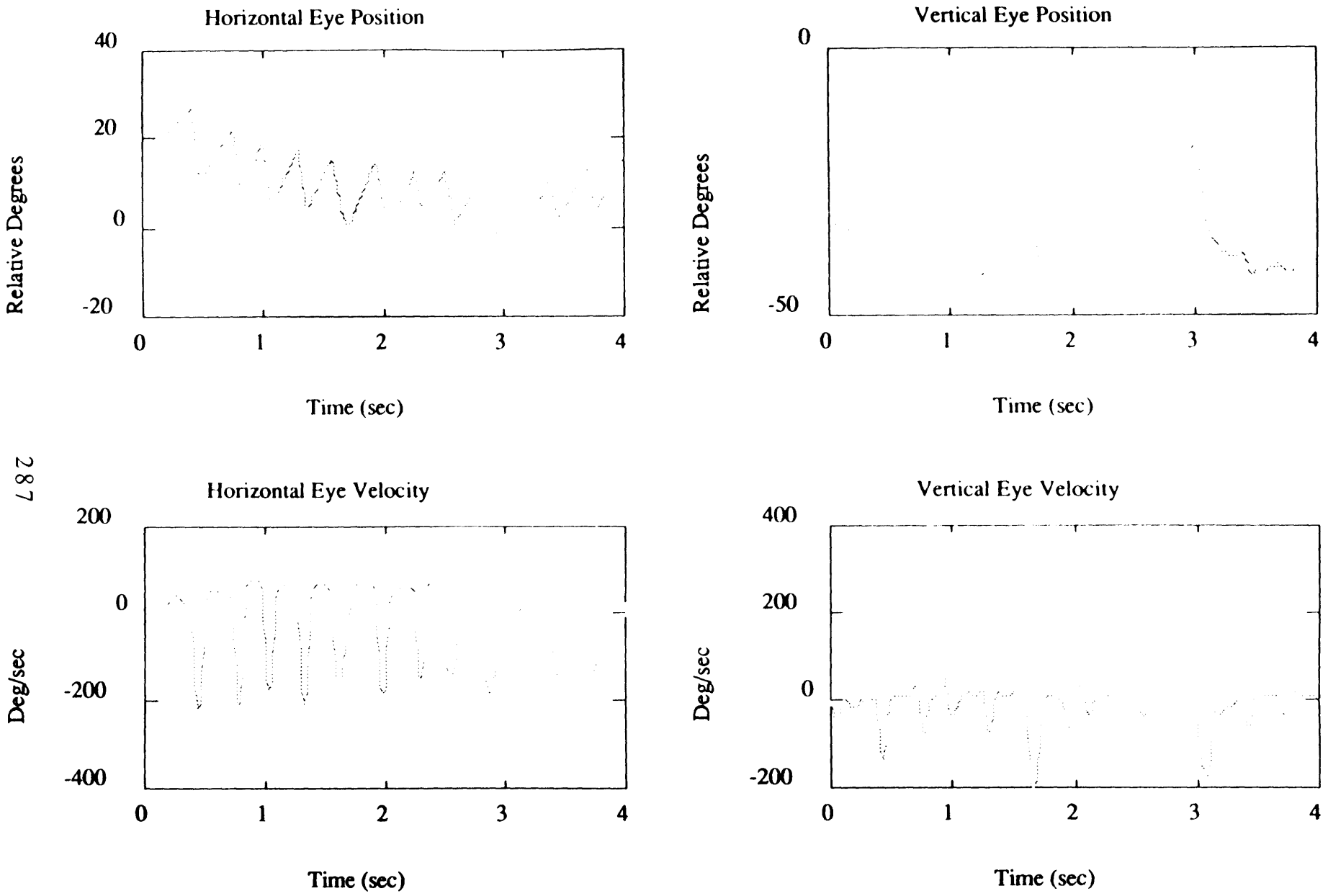


Figure 14.1 Plots of typical nystagmus

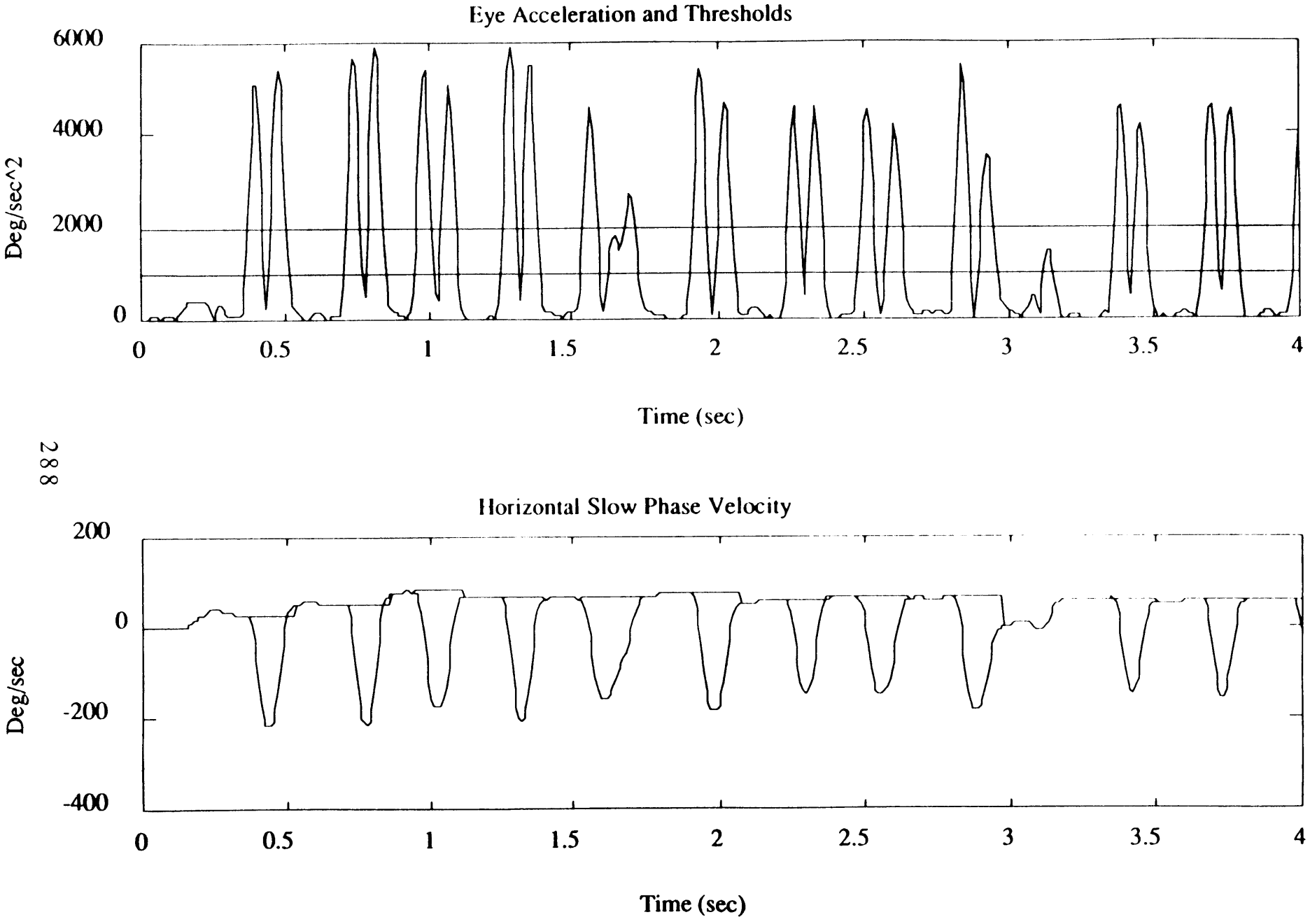


Figure 14.2 Peak theshold 2000, end threshold 1000

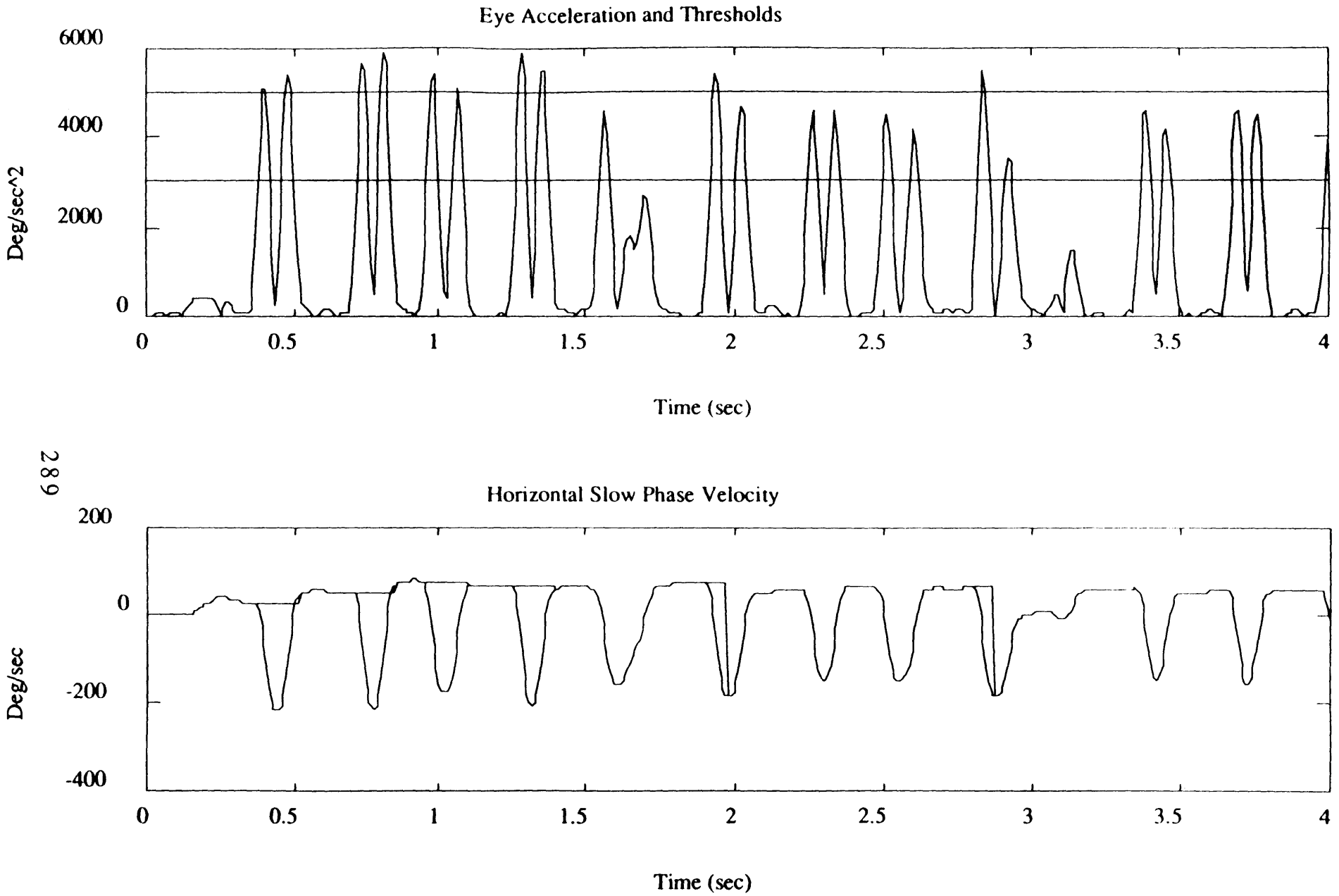


Figure 14.3 Peak threshold 5000, end threshold 3000

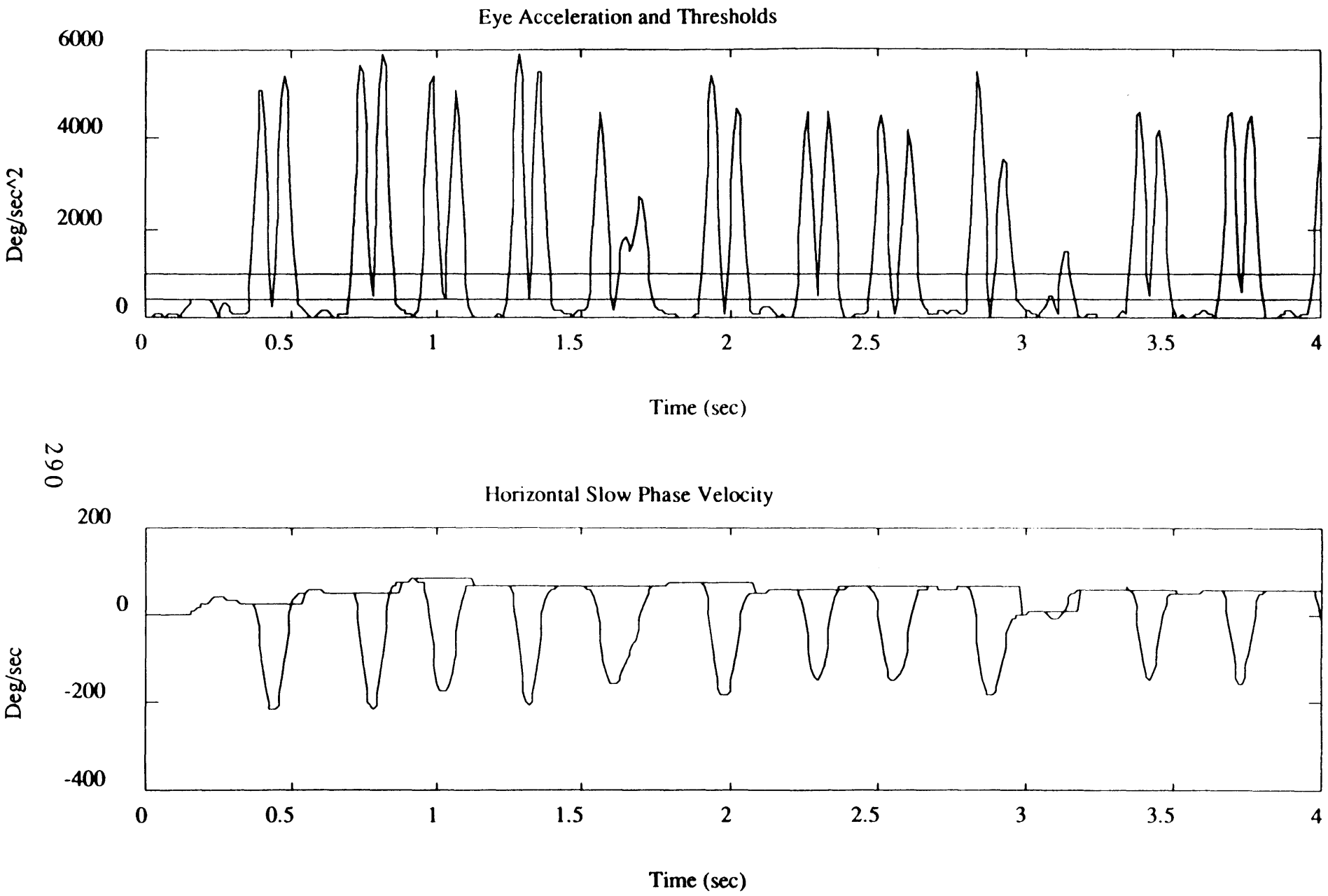


Figure 14.4 Peak threshold 1000, end threshold 400

The performance of the algorithm is strongly dependent upon the choice of the two acceleration thresholds. In Figure 14.2, the thresholds were chosen as 2000 and 1000 deg/sec^2 respectively, as shown by the dotted lines. If the thresholds are set too high, two problems will occur: some of the fast phases will be missed entirely, and the duration of some fast phases will be underestimated. Figure 14.3 shows the results of running the algorithm with thresholds of 5000 and 3000. If the thresholds are set too low (i.e. below the noise level), large sections of the time series may be detected as a single fast phase because the acceleration does not drop below the end threshold. Figure 14.4 shows the results of running the algorithm with thresholds of 1000 and 400.

When a new data set is about to be analyzed, the user may not have a good idea as to what appropriate thresholds would be. Therefore, the user can indicate that the values are unknown by entering negative numbers for them (the acceleration thresholds are input in the *init* script), and estimates will be made of appropriate thresholds on statistical grounds. The mean and median accelerations are first calculated. Since the peak acceleration values are usually an order of magnitude larger than the noise level, the peak values are truncated for estimation purposes; specifically, any acceleration value greater than the sum of the mean and the median is reset to the median value. Then, the standard deviation of the new "acceleration" is determined. The peak threshold is then set at the mean plus two standard deviations, and the end threshold is set at the median plus one standard deviation. This estimation scheme is by no means optimal, but it does attempt to consider the highly skewed distribution of values in the acceleration time series. It should be emphasized that the subsequent detection algorithm operates on the original acceleration time series.

No automated algorithm developed to date is perfect in its detection. Notice in Figure 14.2 that the duration of the first event has been overestimated. After the automated algorithm's detection and interpolation has been optimized by the user, NysA permits interactive manual editing of the slow phase velocity (see Section 13). Missed fast phases can be removed, and incorrectly interpolations can be corrected. In the manual editing stage, the interpolations are first order; whereas the automated algorithm may have difficulty detecting the beginning and end of a fast phase, a human operator should have little trouble.

15. Order Statistics based Fast Phase Removal Algorithm

The *AATM* algorithm is based on an Adaptive Asymmetrically Trimmed Mean order statistic filter implemented as a MatLab mex file. The *AATM* script accepts only one axis of eye position data at a time.

Each eye position series is first filtered using four applications of a Predictive Finite Impulse Response Median Hybrid (PFMH) filter using window sizes of six and ten points. This is done to smooth the nystgamus assuming that nystagmus can be modelled as a piecewise continuous sequence of second order polynomial segments. This filtering preserves the ramp like slow and fast phases of the signal, as well as the sharp transitions between the phases, but attenuates high frequency noise in the signal.

Each eye position series is differentiated to yield the corresponding eye velocity. The differentiation process is implemented as a Remez equal ripple digital filter consisting of a combination of a first order derivative and a low pass filter. the number of points, and the cut-off frequency of the filter depend on the sampling frequency with which the original data was recorded.

The SPV is extracted from the calculated eye velocity using an adaptive asymmetrically

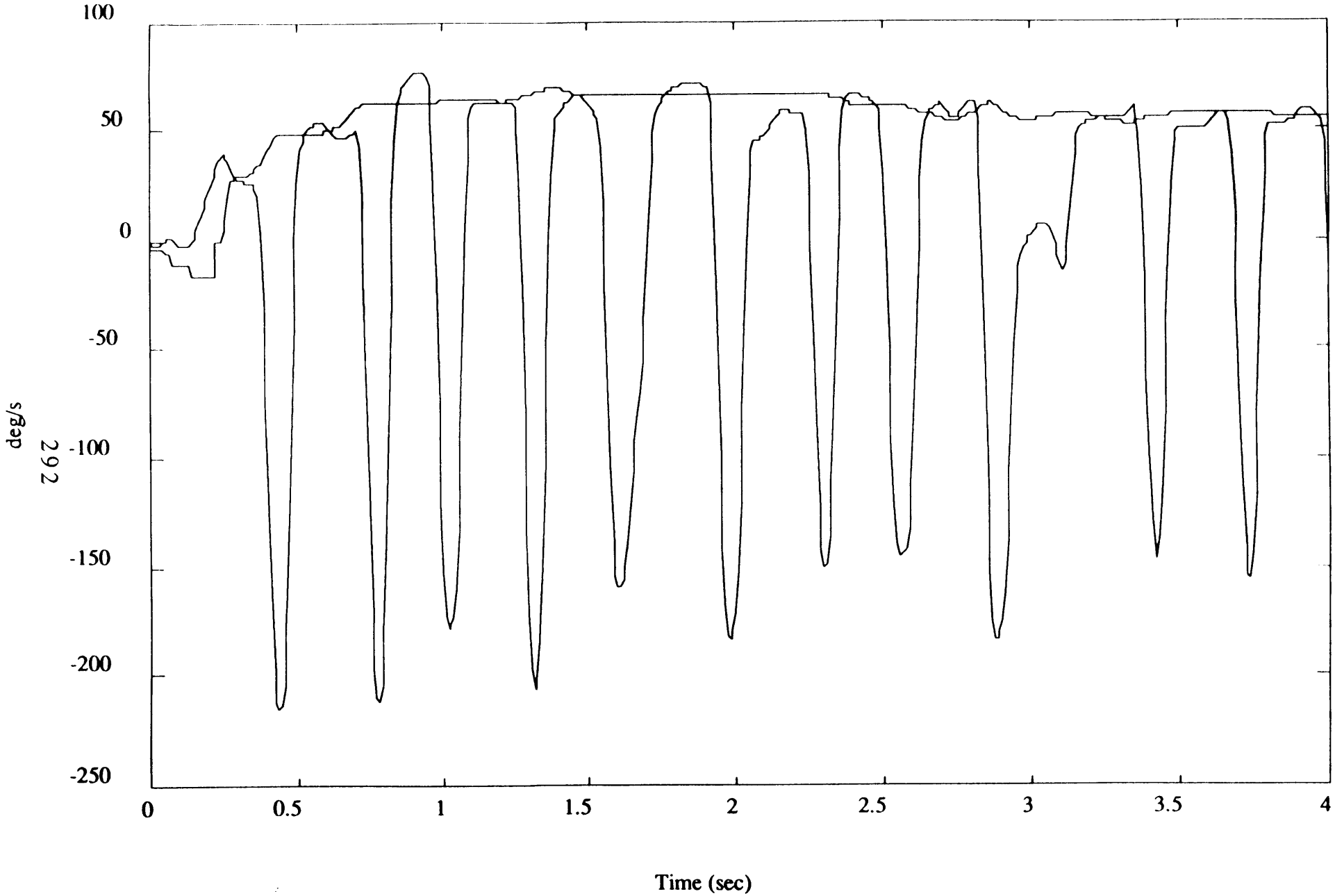


Figure 15.1 Horizontal Slow Phase Velocity -- AATM

trimmed mean filter. This consists of a sliding 120 point (one second at 120Hz) window of data centered on the current point of interest. A subset of data is chosen based on statisitical properties of the velocity window. these subsets are averaged and a selection process chooses only SPV data samples in the average. The averaged SPV is then taken as the output at the current point of interest.

Figure 15.1 shows the *AATM* calculated slow phase velocity for the same data set as used in Section 14. It can be seen that the *AATM* algorithm results in a greatly reduced or eliminated need for manual editing.

16. Closing Comments

The concentration in the development of these scripts has been three-fold (in addition to the scientific integrity of the algorithms). The user should be able to run the scripts in batch mode, so user intervention is to be minimized except in the *calibrate* and *edit_spv* scripts where user input is clearly essential. The various procedures should be capable of being run independently so that a minor change does not require that all of the scripts be rerun. For example, if the acceleration thresholds are changed, only the *init* and *process* stages are essential; the *calibration* and *prep_data* sections should not have to be redone, and manual editing should not be insisted upon. Finally, the scripts should be capable of easy modification and portability to other machines -- this is actually a characteristic of MatLab for the most part, and an effort has been made to take full advantage of it.

This release is simply a current version of a developmental product (insert standard disclaimer here). As such, there are some drawbacks (hopefully, no bugs) which will be improved upon in future versions. Any and all feedback is appreciated.

15. References

Engelken, E.J. and K.W. Stevens (1990). A New Approach to the Analysis of Nystagmus: An Application for Order-Statistic Filters. Aviation, Space, and Environmental Medicine, Vol. 61, No. 9, pp. 859-864.

Merfeld, D.M. (1989). PhD Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology.

Appendix A

Demo

Before the demo can be run, NysA must be installed on the system. In this example, it is to be installed on a removable cartridge disk which is simply called *disk*. The MatLab folder is at the top level. The folders to be added to the MatLab Set Path are therefore *disk:MatLab:NysA_V2.0* and *disk:MatLab:NysA_V2.0:scripts*. Then, *init* is run to set up the NysA path to *disk:MatLab:NysA_V2.0*, accepting the default values for the other parameters. We are now ready to run the *nysa_demo* script.

Included in the *Sample_Data* folder is a data set from a rotating chair experiment, where the subject was seated with the head in an upright position. Five electrodes were used to measure horizontal and vertical eye position. The stimulus was an impulse in acceleration from zero to a constant velocity of 120 °/s for a duration of approximately 60 seconds, followed by a sudden stop. Data is included for the post-rotatory portion only. The two axes of data were sampled at 120 Hz. The files for this data set are *N45.eogh, N45.eogv, N46.eogh,* and *N46.eogv*. We will use this data set as the example in this demo. The *file_specs* script on the distribution disk is set up to set the *data_path* to the proper folder, in this case *disk:MatLab:NysA_V2.0:Sample_Data*, and the file and variable names are set up for this data set. If the *file_specs* script has been modified, a copy of it can be found in *chair_specs*.

The *nysa_demo* script.is quite simple; it merely calls the *init, calibrate, prep_data, process, backup_data,* and *edit_spv* scripts in that order. For the *init* script, the NysA path has already been set up properly. We will specify two axes of information, 120 Hz sampling frequency, and no color monitor. We will leave the acceleration thresholds as *<unknown>* to demonstrate the values that NysA would select, and the resulting performance.

The run code for the calibration script is *N45*. This is encoded in the *file_specs* to give *N45.eogh* and *N45.eogv* as the files containing the input eye position data for a two-axis three-point calibration. Horizontal eye position is specified as the first axis, with vertical eye position as the second axis. The angular deflections are ten degrees in all cases. The cadence for the calibration is centre-right-centre-left-centre-up-centre-down-centre. The horizontal calibration occurs first, between 27 and 31 seconds (see Figure A.1), with the vertical calibration between 33 and 37 seconds. Since the calibration cadence is not repeated, the positive and negative deflections can be specified by selecting a single region (see Figure A.2). However, the zero-deflection point occurs at three locations for each axis, and can therefore be specified by three regions (see Figure A.3). The calibration factors which are calculated will vary slightly, depending on the exact regions which were picked, but should be approximately 0.019 and 0.036 deg/unit for the horizontal and vertical axes respectively.

The files *N46.eogh* and *N46.eogv* contain eye position data from a stimulus run which immediately followed the *N45* calibration. The run code for the *prep_data* script is therefore *N46*. Since this run occurred immediately after the calibration, the calibration factors which were just calculated should be fairly accurate. These values are the defaults, which we accept.

The *prep_data, process,* and *backup_data* scripts now run automatically. Since the acceleration thresholds were indicated as unknown, the calculated thresholds are displayed to the user. Depending on the exact calibration factors, they will be approximately 1850

and 800 respectively.

The data is now ready for manual editing. Since we are primarily interested in the horizontal eye velocity, we select *N46* for the run code and *1* for the axis number. This was the last processed run. As can be seen in the plot of slow phase velocity (see Figure A.4), there are a number of missed saccades. This indicates that perhaps the acceleration thresholds were set too high. Therefore, we will abort this manual editing stage immediately by pressing the escape key, and not saving the edition.

In order to change the acceleration thresholds, the *init* script must be run. Good thresholds are 1100 and 650. Leave all other parameters as before. Since the calibration has already been performed, and the data has already been moved into the NysA folders, we can skip directly to the *process* script, followed by *edit_spv*. Sure enough, the detection is somewhat better now (see Figure A.5).

One obvious "spike" occurs at about six seconds into the plot. By zooming in (see Figure A.6), we can see that two separate events were detected because of a blink (this shows up better on the vertical channel), and that the beginnings and ends of the events were poorly detected. This is an interpolation that should be replaced. First, the two interpolations are selected for deletion (see Figure A.7). Pressing the delete key then replaces these sections with the original eye velocity (see Figure A.8). Now, it is clear that this can be interpolated well as a single event. Selecting points on either side of this complex double spike will yield a proposed interpolation as in Figure A.9. A carriage return will now accept the interpolation, yielding a much more reasonable estimate of slow phase velocity (see Figure A.10).

The remainder of this trace can be edited similarly, producing an estimated slow phase velocity curve like that in Figure A.11. This is good; so, we save this edition. Finally, we must remember that we didn't back up the new data that was obtained from re-running the algorithm with lower acceleration thresholds; therefore, *backup_data* is run. The only data which will have changed in the interim is the estimated slow phase velocity, which we have just edited into a better estimate. While it may seem unnecessary to save it, it is a good general principle to remember to run *backup_data* immediately after *process*.

For comparison purposes, the same data has been processed using the *AATM* script, and included as figure A.12. This demonstrates the reduced need for manual editing when the AATM script is used.

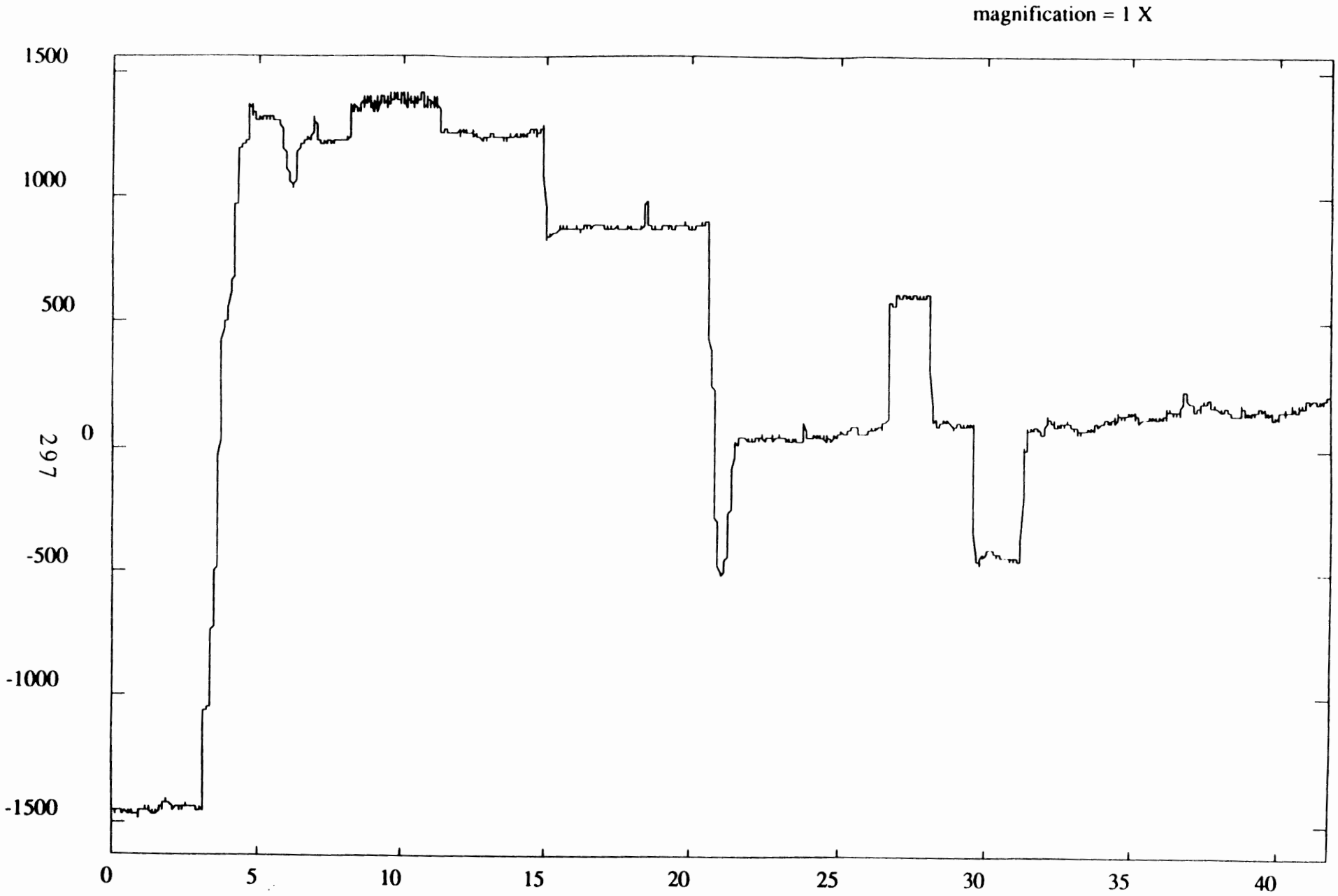

magnification = 1 X
1500
1000
500
0
-500
-1000
-1500
297
0
5
10
15
20
25
30
35
40

Figure A1

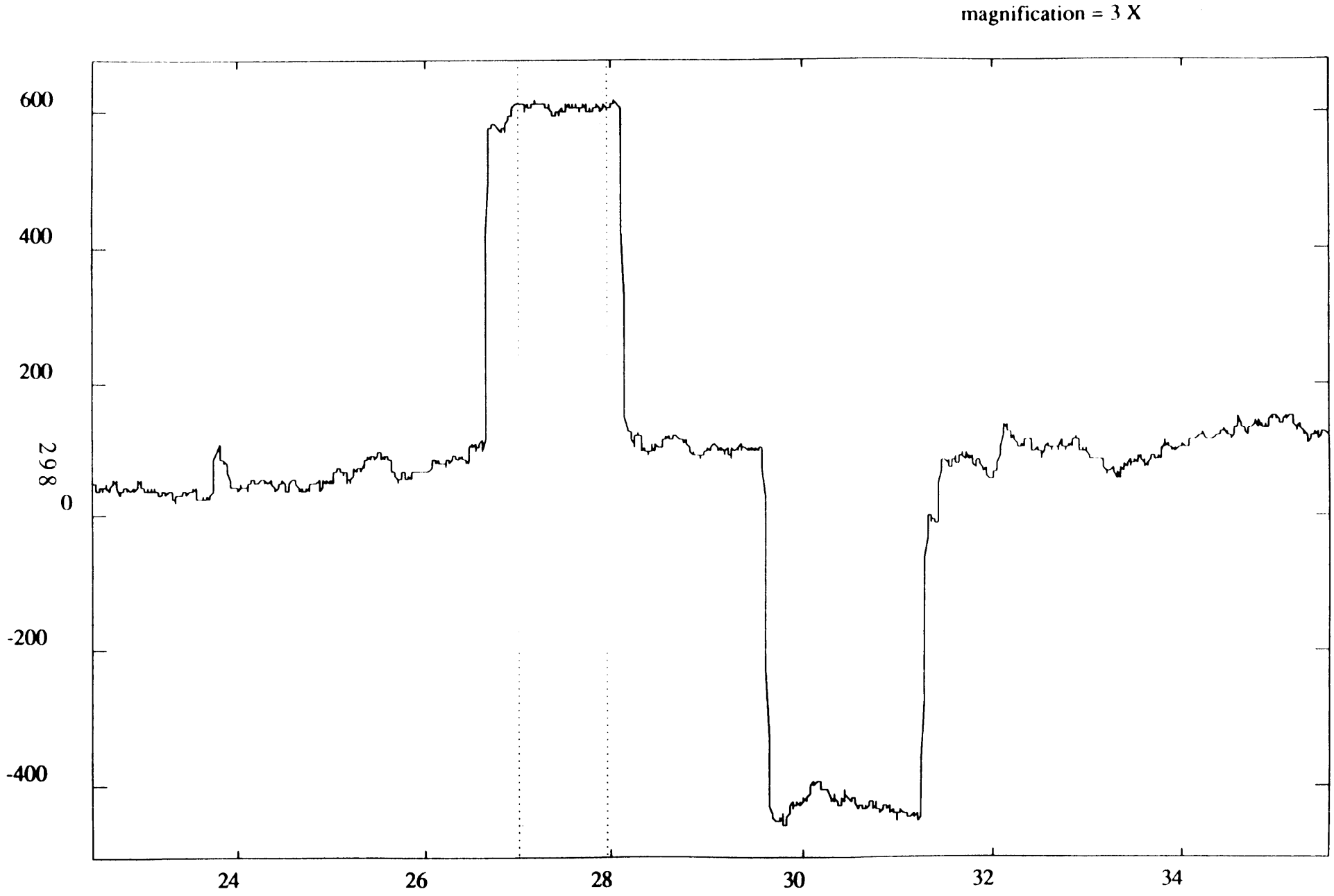


Figure A2

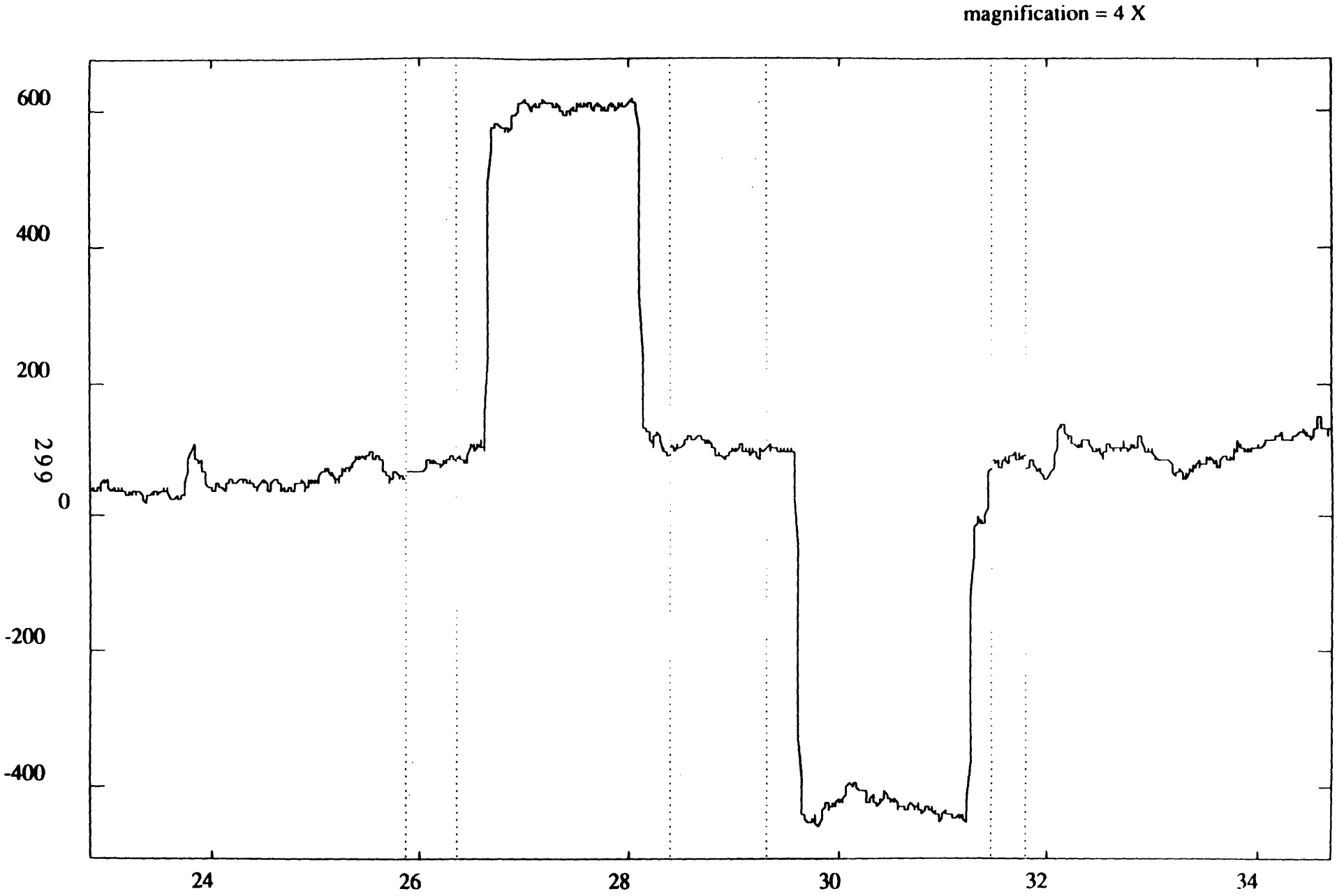
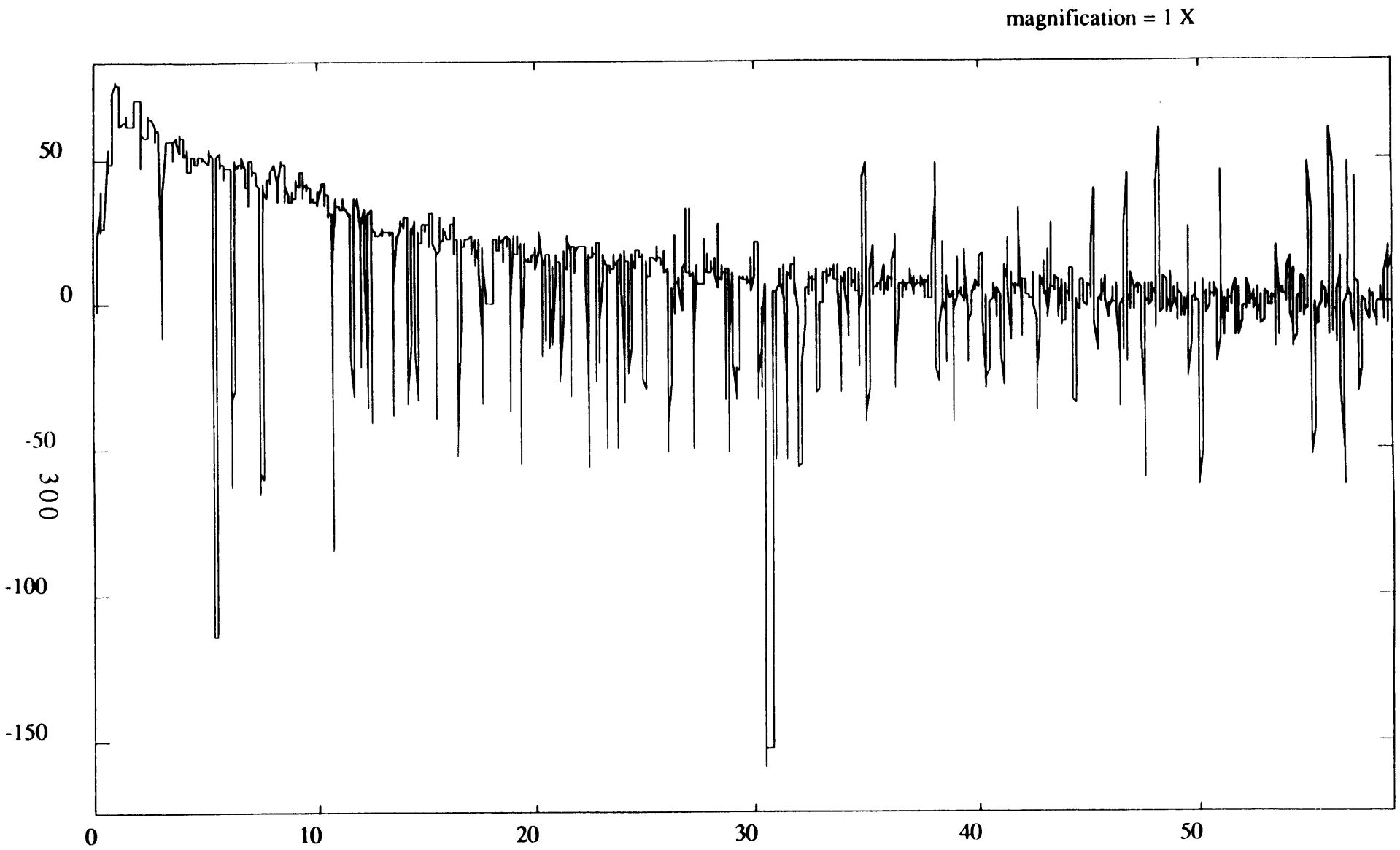
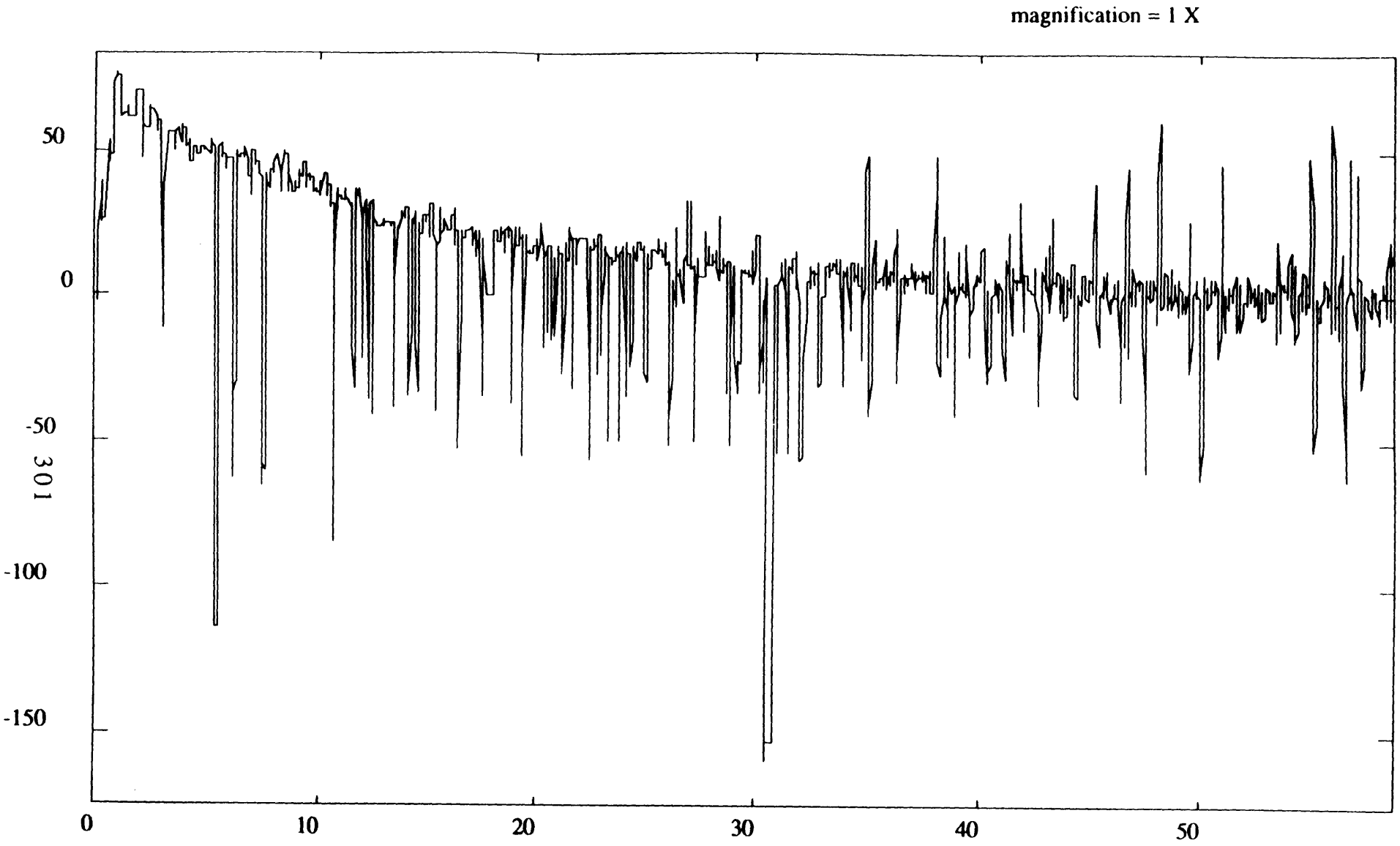


Figure A3



black = SPV

Figure A4



black = SPV

Figure A5

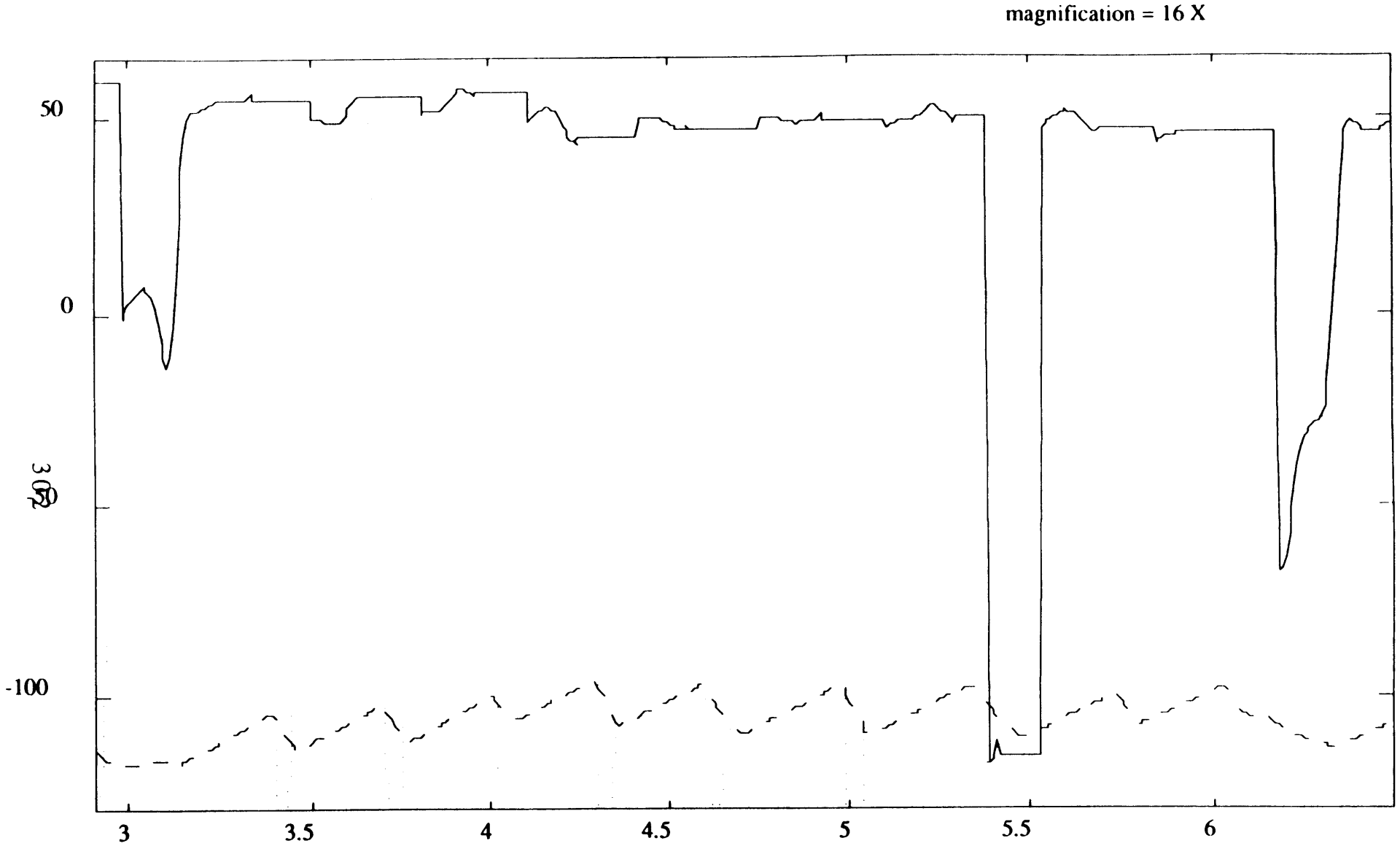
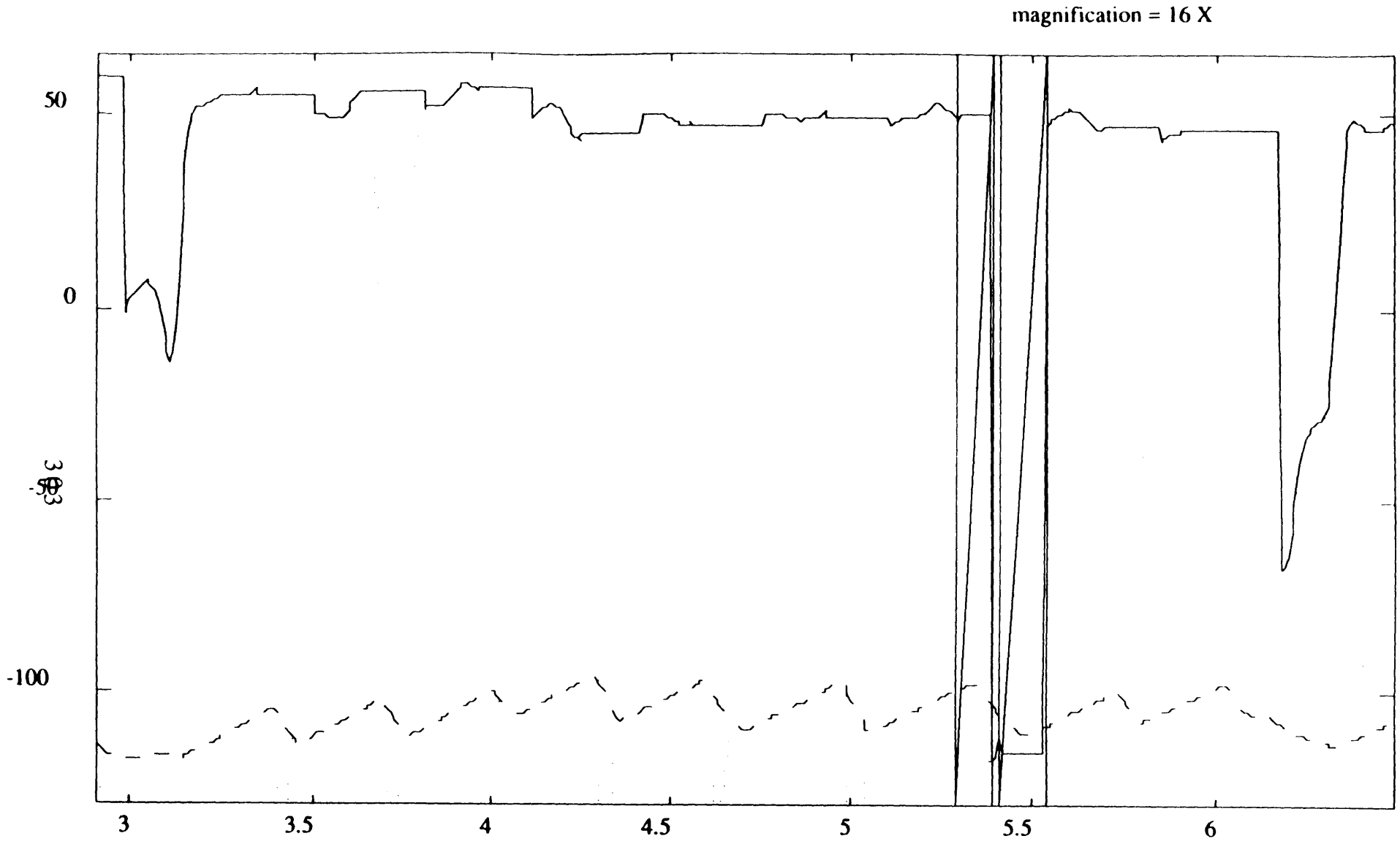

magnification = 16 X
50
0
-100
3
3.5
4
4.5
5
5.5
6
solid = SPV
dotted = raw velocity
dashed = eye position

Figure A6



solid = SPV

dotted = raw velocity

dashed = eye position

Figure A7

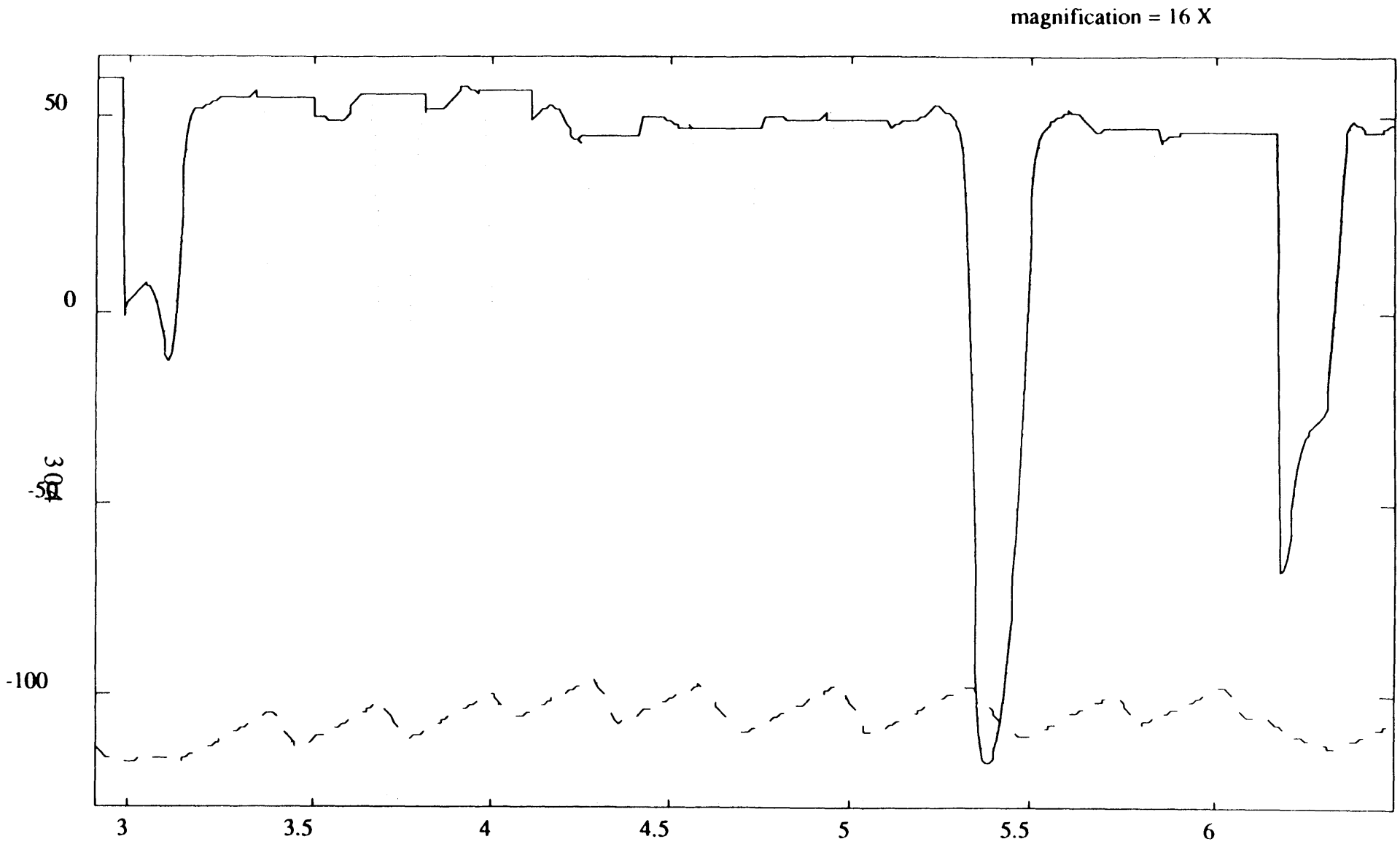

magnification = 16 X
50
0
-50
-100
deg/sec
3
3.5
4
4.5
5
5.5
6
solid = SPV
dotted = raw velocity
dashed = eye position

Figure A8

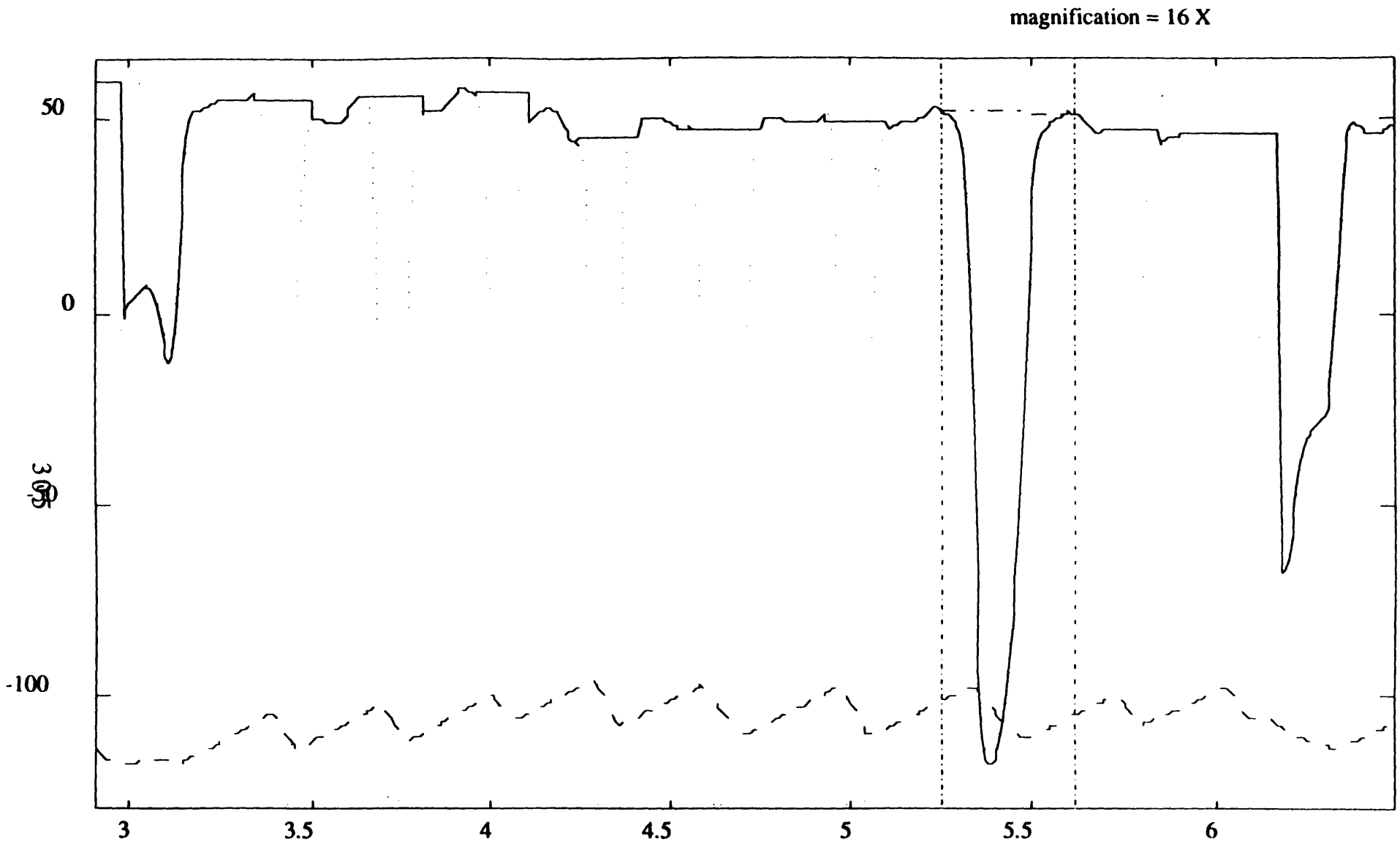
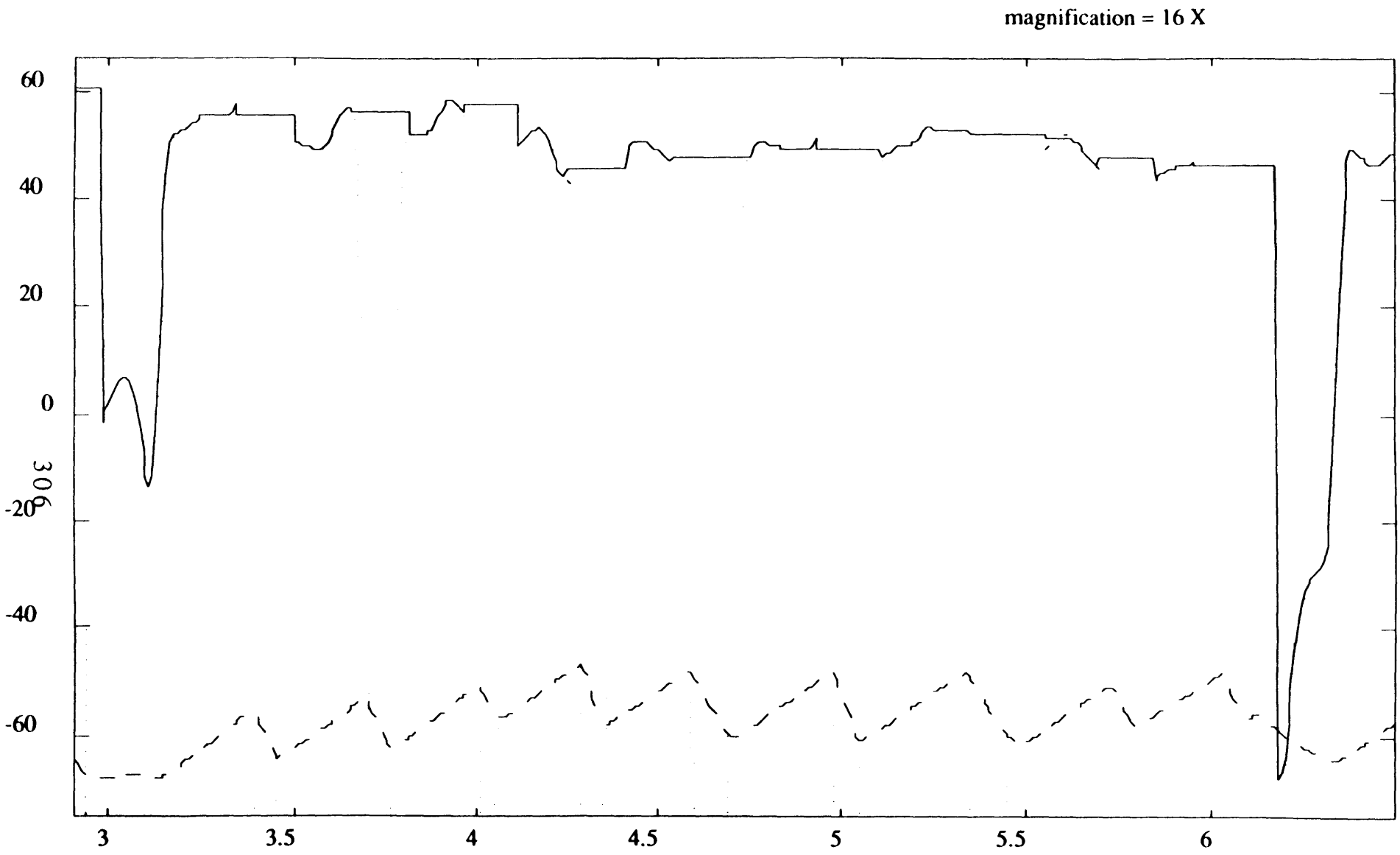
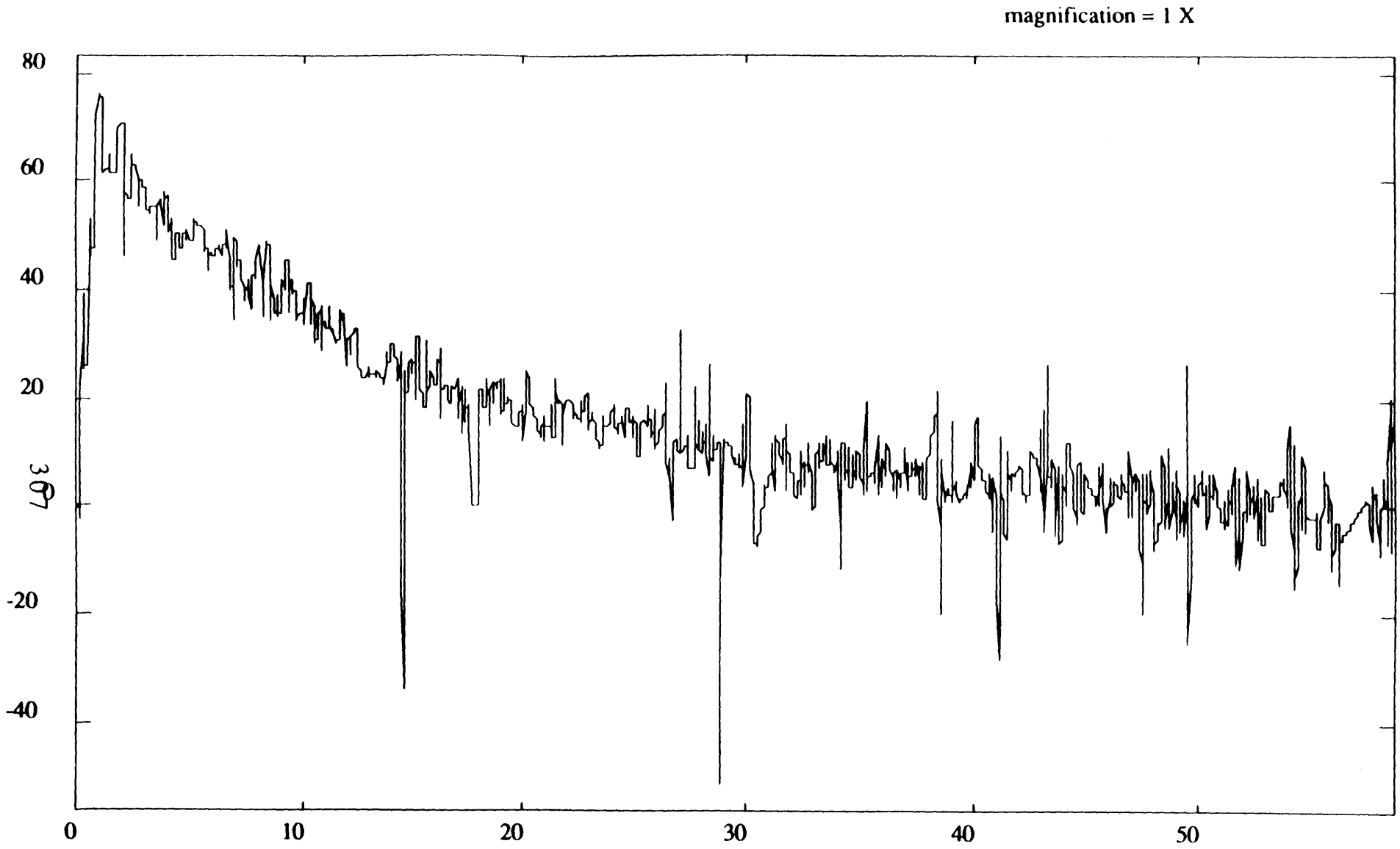

magnification = 16 X
50
0
-100
3
3.5
4
4.5
5
5.5
6
solid = SPV
dotted = raw velocity
dashed = eye position

Figure A9



solid = SPV dotted = raw velocity dashed = eye position

Figure A10



solid = SPV

Figure A11

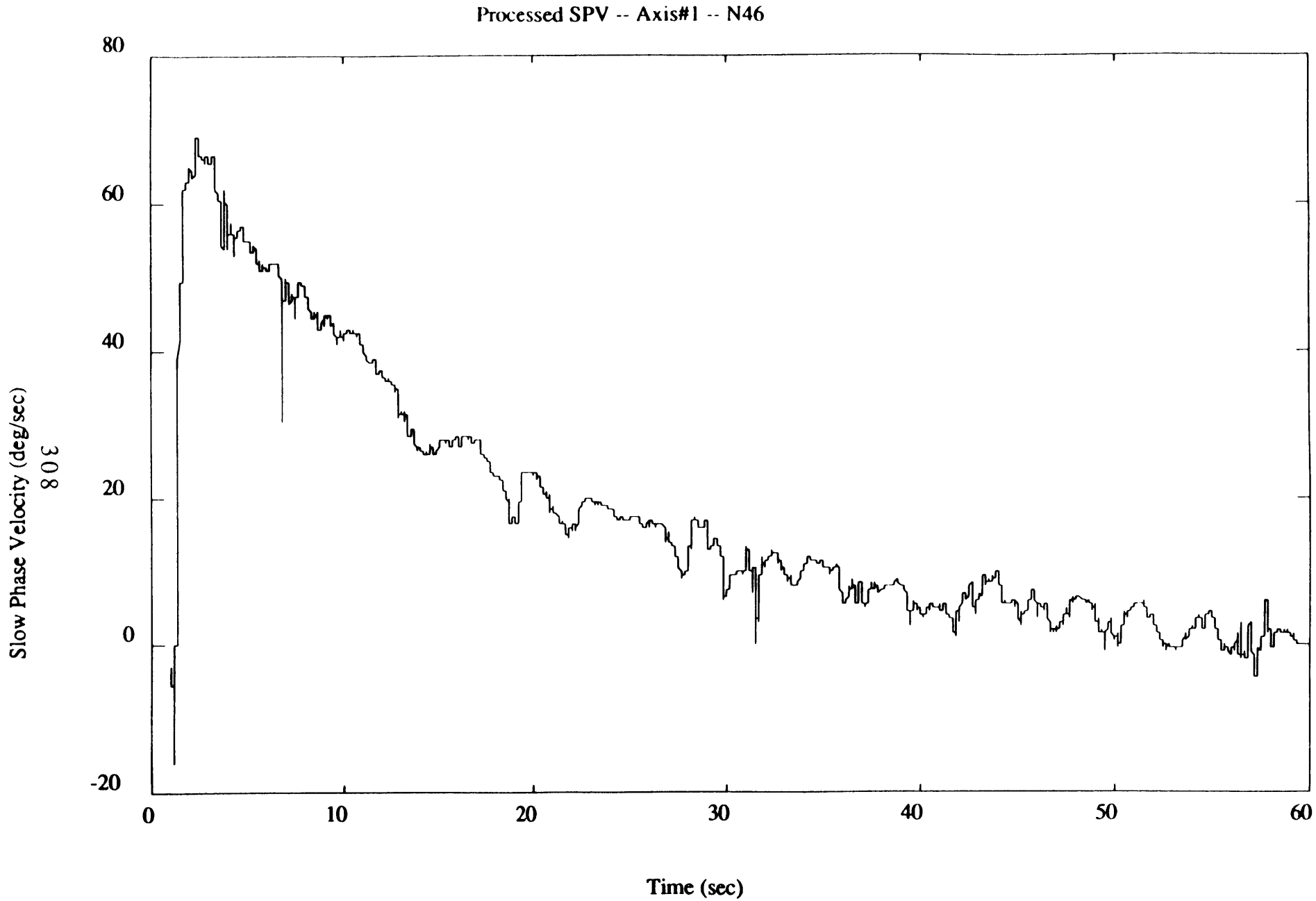


Figure A12 AATM processed slow phase velocity

Appendix B

Sample Data

There are four sets of sample data that are included with NysA_V2.0.

The first set includes files *N45.eogh, N45.eogv, N46.eogh,* and *N46.eogv*. This is the set which is used in the demo in Appendix A. The stimulus was a constant z-axis rotational velocity of 120 °/s for 60 seconds, followed by a sudden stop, with the head upright. The data in *N46.eogh* and *N46.eogv* is horizontal and vertical EOG eye position for the post-rotatory portion, sampled at 120 Hz. *N45.eogh* and *N45.eogv* are horizontal and vertical eye position for a three-point calibration (10° deflection) along each axis. The appropriate *file_specs* for this data set is *chair_specs*.

The second set includes files *HPOS55* and *VPOS55*. The stimulus was a 3 G centrifuge run. The data is horizontal and vertical eye position as recorded by EyeScan, and sampled at 60 Hz. The data is already in degrees. The appropriate *file_specs* for this data set is *eye_specs*.

The third data set includes files *A1A001C1.MAT, A1A001C2.MAT* and *A1A001C3.MAT*. The stimulus was a centrifuge run on squirrel monkeys. The data is horizontal, vertical, and torsional eye position respectively, as recorded by a coil system, and sampled at 200 Hz. The appropriate *file_specs* for this data set is *coil_specs*.

The fourth data set includes file *Y(okn)A*. The stimulus is a windowshade generating opto-kinetic nystagmus. The data is horizontal eye position as measured by EOG, and sampled at 128 Hz. The appropriate *file_specs* for this data set is *okn_specs*.

Appendix C

NysA Script Additions

Additional scripts are available for the NysA package which analyze both axes of data simultaneously. These scripts more effectively extract invalid readings by altering the vertical data in the regions identified as horizontal fast phases. There is also a script to allow manual editing and analysis of both horizontal and vertical data simultaneously. These scripts are modifications of the original NysA scripts, and go by the following names;

file_specs_sls
switch
calibrate_sls
batch_sls
edit_spv_dual.

These scripts are available upon request.

Appendix D

Summary of Editing Commands

Calibrate

Action	Command(s)
Zoom In	up arrow
Zoom Out	down arrow
Scroll Right full screen	page up
1/4 screen	left arrow
Scroll Left full screen	page down
1/4 screen	right arrow
Fast Zoom	press " . " mouse click on left and right borders
Reset Magnification to 1	"0"
Pick Cal. Points	mouse click on left and right borders
Exit	<CR>

Edit_Spv

Action	Command(s)
Zoom In	up arrow
Zoom Out	down arrow
Scroll Right full screen	page up
1/4 screen	left arrow
Scroll Left full screen	page down
1/4 screen	right arrow
Fast Zoom	press " . " mouse click on left and right borders
Reset Magnification to 1	"0"
Select a Region	pick left and right borders with mouse click
Select an Interpolation	mouse click inside the region
De-select a Region or Interpolation	mouse click inside the highlighted region or interpolation
Delete all Selected Interpolations	<Del>
Interpolate Across all Selected Regions	<CR>
Exit	<Esc>

Appendix F

AATM.c Source Code

Notes on AATM Program
D. Balkwill 12/22/91

AATM is a Think C implementation of the algorithm proposed by Engelken and Stevens (1990) for estimation of the slow phase velocity (SPV). The name of the program is a bit of a misnomer since the AATM portion is itself only a part of the overall algorithm. This implementation was originally done for the purposes of processing rotating chair data from the SLS-1 mission. However, a number of modifications have made it very flexible to a number of different data types. It should be linked with the *unix* and *ANSI-881* libraries, with the 68020 and 68881 options set on the compilation.

Basic Algorithm

The basic algorithm consists of two separate stages, each of which is a non-linear order statistic (OS) filter. The first type of filter is PFMH (predictive FIR mean hybrid) which is used to enhance the eye position signal by reducing signal noise due to EMI, quantization, or spike artifacts, and by sharpening the corners of the nystagmus. Estimates for the eye position at a given sample point are determined by the eye position at a number (N) of sample points before and after that point. Forward estimates are determined by applying weighting coefficients to the N preceding samples; backward estimates are determined by applying those same weighting coefficients to the N succeeding samples. This program calculates two forward estimates based upon N=6 and N=10 preceding points, and the two corresponding backward estimates. The median of these five points (four estimates andthe original data point) is taken to be the new eye position.

The PFMH filters operate on the basis of "root" signals, which are invariant under application of the filter. The difference between the actual eye position signal and a root signal is "noise" and is reduced with each pass of the filter. For the rotating chair data, horizontal eye position is of primary importance. Therefore, linear weighting coefficients were used, corresponding to the first-order (linear) root signals typical of horizontal vestibular nystagmus.

The second type of OS filter is AATM (adaptive asymmetrically trimmed-mean) which is used to estimate the slow phase velocity from the raw eye velocity. It operates on a sliding window principle such that the SPV at a given sample point is calculated from one second of velocity sample values, with the window centred at the point of interest. The underlying assumption is that, during nystagmus, the eye spends more time in slow phase movements than fast phase; therefore, the velocity values will be mostly centred around some value corresponding to the SPV, with a smaller number of values centred around the fast phase velocity (FPV) which will be of the opposite sign, and with a few intervening samples. The velocity values in the window are sorted, a few samples at the beginning and end of the list are discarded, and the skewness S_β of the remaining distribution is calculated. An index into the sorted list is calculated by shifting from the centre by an amount proportional to S_β; this index is analogous to the mode or peak of the distribution. The mean of a number of points about this index is taken as the estimate of the SPV at the centre of this window.

Three dimensionless parameters (α, β, and γ) can be specified by the user to control certain attributes of this procedure. α controls the number of data points which are averaged to yield the SPV estimate; β controls the range of values which are included

in the skewness calculation; γ controls the number of points by which the median may be shifted to yield the mode. The default parameters used by Engelken and Stevens were 0.44, 0.12, and 0.186 respectively. A study by Engelken, Stevens, and Enderle (1991) showed that the performance of the algorithm was relatively insensitive to changes in these parameters. For SLS-1 rotating chair data, γ was increased to 0.4 to reflect the substantially higher beat frequencies and peak SPV.

A more detailed description of the PFMH and AATM filters can be found in the papers by Engelken and Stevens or in my thesis (Section 4.4).

Overview

AATM uses the same *file_specs* concept as *NysA* (see Appendix D), and is incorporated in a group of *#defines* near the beginning of the program.. Any experimental run (stimulus or calibration) is given a "run code" which tags all the data files corresponding to that run. The actual file and variable names can be controlled by the user such that variable names are consistent across all runs, and file names vary only in the run code itself. Here, the variable and file name for horizontal eye position are *eogh* and *#.eogh* respectively. The number sign is replaced by the run code to construct the actual file name. Similarly, the vertical eye position (*eogv*), horizontal and vertical eye velocity (*velh* and *velv*), horizontal and vertical AATM slow phase velocity (*aspvh* and *aspvv*), and horizontal and vertical PFMH-filtered eye position (*osh* and *osv*) are specified. The file names are constructed by calls to the *file_name* (similar to the NysA script of the same name) from the *create_output_file* routine.

Notice that there is no *data_path* included in the program. Therefore, unless a few tricks are known about the way paths are specified in the Macintosh (hint: colons separate levels of hierarchy), the AATM program must be in the same folder as the data which is to be processed.

The user inputs a run code and two calibration factors (horizontal and vertical). The horizontal and vertical data are treated similarly but separately since AATM is a single-axis algorithm. Memory space is allocated dynamically so that the eye position data can be loaded into a single properly-sized chunk of memory. Notice that this puts a limit of 32K on the number of samples which can be processed by AATM, due to the limitations of C in indexing arrays. The buffer which is allocated depends upon the data type (signed or unsigned, real or integer, long or short) under which the eye position is saved. Memory buffers are also allocated for the eye velocity (*vel_buffer*), SPV (*spv_buffer*), and a standardized eight-byte real buffer for the filtered eye position (*pos_buffer*) independent of the input data type.

The eye position data is loaded and then processed through two passes of the PFMH filter described above. The filtered eye position is saved on the disk, and then differentiated to yield the eye velocity. This differentiation filter is identical to that used in NysA and is incorporated in the *velA* and *velB* definitions near the beginning of the program. Notice that the sample rate dependency is included entirely in *velA*; *velB* is invariant. This velocity is saved and then passed to the AATM routine, which calculates the SPV. The SPV is then saved, the memory buffers are cleared, and the files are closed.

If at any time the program attempts to overwrite a file which already exists, the user is prompted for permission; if this is not granted, the old file remains on the disk and the program ceases processing of that channel of data.

Batch Mode

AATM loops for as many runs as the user may wish to process. Alternatively, if there are a large number of runs which are ready for processing, AATM can be run in batch mode. This involves creating an ASCII batch file containing the run codes and calibration factors. A run code must appear on the first line, a horizontal calibration factor on the second, and a vertical calibration factor on the third. This is to be followed by another run's trio of run code and calibration factors; no blank lines should exist. There are many ways to create such a file, one of the most common being Microsoft Word. However, a file saved in "normal" format in Word has a header 256 bytes, of which the first byte is 0xFE. An extra check is made to see if the batch file is of this format. If a file is saved as "text only" in Word, this problem does not arise.

Modifications

Different experiments may need to perform a few slight modifications to the code. If the sampling rate is different (e.g. 60, 128, or 200 Hz), the *SAMPLE_RATE* definition at the beginning of the program needs to be altered. This will be reflected in the velocity filter so that the velocity is correctly calculated, and in the maintenance of a one-second window for the AATM SPV estimation algorithm. However, the lengths of the PFMH filters are not automatically altered. For 60 Hz data, *N1* and *N2* should be changed to 4 and 8 to reflect the lower sampling frequency. For 200 Hz data, *N1* and *N2* should be changed to 10 and 15 to reflect the higher sampling frequency. For sampling rates close to 120 Hz (e.g. 100, 128), the given values should suffice. Ideally, the lengths of these filters should be time-invariant and should reflect the durations of the fast and slow phase eye movements. In practice however, they should not be reduced too far, or else they will be essentially useless, and perhaps detrimental.

The most common type of modification would be the desire for a different file and naming convention. Such changes need only be made to the *#define*s at the beginning of the program, following the guidelines described in the NysA manual.

It is also quite possible that a different A/D board could be used which has an entirely different range and resolution. This should require only changes in the *A2D_SCALE* parameter. There is no step in the program for removing an offset from the position data, since only the differential velocity was of interest; however, this can be implemented easily in the *calc_vel_data* routine.

Other parameters which are implemented as *#define*s to permit easy alteration by the user are the α, β, and γ parameters for the AATM algorithm, and the data type of the output MatLab files. The velocity filter coefficients can also be easily changed to instead use a filter with a different response.

```
/* AATM.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unix.h>
#include <fcntl.h>

#define FALSE 0
#define TRUE  1

#define MAX_LINE_LENGTH 81

#define ALPHA 0.44
#define BETA  0.12
#define MU    0.4

#define NUM_BUFFER 4000

#define SAMPLE_RATE 120

/*
  MatLab types for input data buffers -- see MatLab manual for more detailed explanation
*/
#define USI 4          /* unsigned short integer (16-bit) */
#define SI 3           /* signed short integer (16-bit) */
#define LI 2           /* signed long integer (32-bit) */
#define R4 1           /* signed real (32-bit) */
#define SD 0           /* signed short double (64-bit) */
#define COL 0          /* column orientation */
#define ROW 1          /* row orientation */

#define USI_COL_TYPE      (1000 + COL * 100 + USI * 10)
#define USI_ROW_TYPE      (1000 + ROW * 100 + USI * 10)
#define SI_COL_TYPE       (1000 + COL * 100 + SI * 10)
#define SI_ROW_TYPE       (1000 + ROW * 100 + SI * 10)
#define LI_COL_TYPE       (1000 + COL * 100 + LI * 10)
#define LI_ROW_TYPE       (1000 + ROW * 100 + LI * 10)
#define R4_COL_TYPE       (1000 + COL * 100 + R4 * 10)
#define R4_ROW_TYPE       (1000 + ROW * 100 + R4 * 10)
#define SD_COL_TYPE       (1000 + COL * 100 + SD * 10)
#define SD_ROW_TYPE       (1000 + ROW * 100 + SD * 10)

/*
  file and variable name specifications, equivalent to NysA file_specs
*/
#define EOGH_VAR "eogh"
#define EOGV_VAR "eogv"
#define VELH_VAR "velh"
#define VELV_VAR "velv"
#define ASPVH_VAR "aspvh"
#define ASPVV_VAR "aspvv"
#define OSH_VAR "osh"
#define OSV_VAR "osv"

#define EOGH_FILE "#.eogh"
#define EOGV_FILE "#.eogv"
#define VELH_FILE "#.velh"
```

```
#define VELV_FILE "#.velv"
#define ASPVH_FILE "#.aspvh"
#define ASPVV_FILE "#.aspvv"
#define OSH_FILE "#.osh"
#define OSV_FILE "#.osv"


/* MatLab file header format */
typedef struct {
   long type;
   long mrows;
   long ncols;
   long imagf;
   long namlen;
   } Fmatrix;
Fmatrix header;


/* input data buffers */
int input_data_type;
short int *si_buffer;
long int *li_buffer;
short double *sd_buffer;
float *r4_buffer;

/* output data buffers */
short double *pos_buffer, *vel_buffer, *spv_buffer;
int pos_handle, vel_handle, spv_handle, os_handle;
FILE *pos_fptr, *vel_fptr, *spv_fptr;
#define OUT_TYPE SD_COL_TYPE        /* output data files are all 8-byte real */


/* required character strings */
char run_code[MAX_LINE_LENGTH]; /* run code for data files */
char in_line[MAX_LINE_LENGTH];              /* user input */
char matlab_name[MAX_LINE_LENGTH];          /* name of MatLab variable */


/* batch mode variables */
char batch_file_name[MAX_LINE_LENGTH];
int batch_mode = FALSE;
int batch_overwrite = FALSE;
FILE *batch_fptr;


/* number of bytes and samples input by a "read" call */
size_t in_bytes, in_samples;


/* number of samples in the overall data file */
size_t num_samples;


/* coefficients for digital differentiation filter, as taken from NysA */
#define FILTER_LENGTH 9
#define velA  (1 / (double)SAMPLE_RATE)
double velB[FILTER_LENGTH] =
    {0.0332, 0.0715, 0.0678, 0.0522, 0, -0.0522, -0.0678, -0.0715, -0.0332};
```

```
/* horizontal and vertical calibration factors */
double hcal, vcal;


/***********************************************
        input buffer memory allocation macros
***********************************************/

/* allocate memory for short double (8-byte) buffer */
#define ALLOC_SD_BUFFER(BUF,NUM) \
{\
(BUF) = (short double *)malloc((NUM) * sizeof(short double));\
if (!(BUF)) {\
printf("Out of memory on buffer allocation.\n");\
goto done;\
}\
}

/* allocate memory for  4-byte real buffer */
#define ALLOC_R4_BUFFER(BUF,NUM) \
{\
(BUF) = (float *)malloc((NUM) * sizeof(float));\
if (!(BUF)) {\
printf("Out of memory on buffer allocation.\n");\
goto done;\
}\
}

/* allocate memory for short int (2-byte) buffer */
#define ALLOC_SI_BUFFER(BUF,NUM) \
{\
(BUF) = (short int *)malloc((NUM) * sizeof(short int));\
if (!(BUF)) {\
printf("Out of memory on buffer allocation.\n");\
goto done;\
}\
}

/* allocate memory for long integer (4-byte) buffer */
#define ALLOC_LI_BUFFER(BUF,NUM) \
{\
(BUF) = (long int *)malloc((NUM) * sizeof(long int));\
if (!(BUF)) {\
printf("Out of memory on buffer allocation.\n");\
goto done;\
}\
}


/***********************************************
        input buffer read  macros
***********************************************/

/* read in short double (8-byte) data buffer */
#define READ_SD_BUFFER(HNDL,BUF,NUM) \
{\
```

```
in_bytes = read((HNDL),(char *)(BUF),((NUM) * sizeof(short double)));\
in_samples = in_bytes / sizeof(short double);\
printf("read %ld samples\n",in_samples);\
}

/* read in 4-byte real data buffer */
#define READ_R4_BUFFER(HNDL,BUF,NUM) \
{\
in_bytes = read((HNDL),(char *)(BUF),((NUM) * sizeof(float)));\
in_samples = in_bytes / sizeof(float);\
printf("read %ld samples\n",in_samples);\
}

/* read in short integer (2-byte) data buffer */
#define READ_SI_BUFFER(HNDL,BUF,NUM) \
{\
in_bytes = read((HNDL),(char *)(BUF),((NUM) * sizeof(short int)));\
in_samples = in_bytes / sizeof(short int);\
printf("read %ld samples\n",in_samples);\
}

/* read in long integer (4-byte) data buffer */
#define READ_LI_BUFFER(HNDL,BUF,NUM) \
{\
in_bytes = read((HNDL),(char *)(BUF),((NUM) * sizeof(long int)));\
in_samples = in_bytes / sizeof(long int);\
printf("read %ld samples\n",in_samples);\
}


/***********************************************
        output buffer write  macros
***********************************************/

#define WRITE_SD_BUFFER(HNDL,BUF,NUM) \
{\
printf("writing %ld samples\n",(size_t)(NUM));\
write((HNDL),(char *)(BUF),(NUM)*sizeof(short double));\
}



main()
{
  int open_input_file(), create_output_file(), get_file_parameters();
  void get_cal_factor(), load_pos_data(), write_matlab_header(), AATM(), check_batch_mode();
  void OS_lin2(), calc_vel_data(), save_vel_data(), save_os_data();

  check_batch_mode();

  while (get_run_code()) {

    get_cal_factors();

    if (pos_handle = open_input_file(EOGH_FILE)) {

      if (vel_handle = create_output_file(VELH_FILE)) {
```

```
if (spv_handle = create_output_file(ASPVH_FILE)) {

    if (get_file_parameters(pos_handle,EOGH_VAR)) {

        /* allocate memory buffer for input data file, dependent upon data type */
        switch (input_data_type) {
          case SI:
                ALLOC_SI_BUFFER(si_buffer,(size_t)(num_samples + SAMPLE_RATE))
            break;
          case LI:
                ALLOC_LI_BUFFER(li_buffer,(size_t)(num_samples + SAMPLE_RATE))
            break;
          case R4:
                ALLOC_R4_BUFFER(r4_buffer,(size_t)(num_samples + SAMPLE_RATE))
            break;
          case SD:
                ALLOC_SD_BUFFER(sd_buffer,(size_t)(num_samples + SAMPLE_RATE))
            break;
          }
        /* allocate memory buffer for output data files */
        ALLOC_SD_BUFFER(pos_buffer,(size_t)(num_samples + SAMPLE_RATE))
        ALLOC_SD_BUFFER(vel_buffer,(size_t)(num_samples + SAMPLE_RATE))
        ALLOC_SD_BUFFER(spv_buffer,NUM_BUFFER)

        /* load and PFMH-filter the eye position data */
        load_pos_data();
        OS_lin2();
        OS_lin2();

        /* save PFMH-filtered eye position */
        os_handle = create_output_file(OSH_FILE);
        save_os_data(OSH_VAR);
        close(os_handle);

        /* calculate and save eye velocity data, and pass to AATM routine */
        calc_vel_data(hcal);
        save_vel_data(VELH_VAR);
        write_matlab_header(spv_handle,ASPVH_VAR);
        AATM();

        /* free up space from memory buffers */
        switch (input_data_type) {
          case SI:
            free(si_buffer);
            break;
          case LI:
            free(li_buffer);
            break;
          case R4:
            free(r4_buffer);
            break;
          case SD:
            free(sd_buffer);
            break;
          }
        free(pos_buffer);
        free(vel_buffer);
        free(spv_buffer);
```

```
            }

            }
        }
        }
if (pos_handle = open_input_file(EOGV_FILE)) {

   if (vel_handle = create_output_file(VELV_FILE)) {

      if (spv_handle = create_output_file(ASPVV_FILE)) {

         if (get_file_parameters(pos_handle,EOGV_VAR)) {

            switch (input_data_type) {
               case SI:
                     ALLOC_SI_BUFFER(si_buffer,(size_t)(num_samples + SAMPLE_RATE))
                 break;
               case LI:
                     ALLOC_LI_BUFFER(li_buffer,(size_t)(num_samples + SAMPLE_RATE))
                 break;
               case R4:
                     ALLOC_R4_BUFFER(r4_buffer,(size_t)(num_samples + SAMPLE_RATE))
                 break;
               case SD:
                     ALLOC_SD_BUFFER(sd_buffer,(size_t)(num_samples + SAMPLE_RATE))
                 break;
               }
            ALLOC_SD_BUFFER(pos_buffer,(size_t)(num_samples + SAMPLE_RATE))
            ALLOC_SD_BUFFER(vel_buffer,(size_t)(num_samples + SAMPLE_RATE))
            ALLOC_SD_BUFFER(spv_buffer,NUM_BUFFER)

            load_pos_data();
            OS_lin2();
            OS_lin2();

            os_handle = create_output_file(OSV_FILE);
            save_os_data(OSV_VAR);
            close(os_handle);

            calc_vel_data(vcal);
            save_vel_data(VELV_VAR);
            write_matlab_header(spv_handle,ASPVV_VAR);
            AATM();

            switch (input_data_type) {
               case SI:
                 free(si_buffer);
                 break;
               case LI:
                 free(li_buffer);
                 break;
               case R4:
                 free(r4_buffer);
                 break;
               case SD:
                 free(sd_buffer);
                 break;
```

```
                    }
                free(pos_buffer);
                free(vel_buffer);
                free(spv_buffer);

                }

                }
        }
        }
    }

done:
  /* close data files if necessary */
  if (pos_handle > 0)
    close(pos_handle);
  if (vel_handle > 0)
    close(vel_handle);
  if (spv_handle > 0)
    close(spv_handle);
}


/*****************************************************************************
  This routine checks to see if the user wishes to operate in batch mode. If yes, the user must enter the
  name of the batch file containing the list of run codes and calibration factors, as well as indicating
  whether or not pre-existing data files should be overwritten. If this file exists, the run codes and cal
  factors are read from this file. If any of these conditions is not met, the program runs in interactive
  mode, wherein the user is prompted for the run codes and cal factors one at a time. It is called from
  the mainline.
*****************************************************************************/
void check_batch_mode()
{
    int i, j, l;

    batch_mode = FALSE;

    printf("Do you want to operate in batch mode (y/[n]) ? ");
    gets(in_line);
    if (strlen(in_line) > 0) {
        for (i = 0; in_line[i] <= ' '; i++);
        if ((in_line[i] == 'y') || (in_line[i] == 'Y')) {
            printf("Enter name of file containing run codes and calibration factors: ");
            gets(in_line);
            l = strlen(in_line);
            i = 0;
            while (in_line[i] <= ' ') {
                i++;
                if (i >= l) break;
                }
            if (i < l) {
                j = i;
                while (in_line[j] > ' ') {
                    j++;
                    if (j == l) break;
                    }
                strncpy(batch_file_name,&(in_line[i]),j-i);
                batch_file_name[j-i] = '\0';
```

```
                batch_fptr = fopen(batch_file_name,"rt");
                if (batch_fptr != NULL) {
                    batch_mode = TRUE;
                    printf("Do you want to overwrite all existing files (y/[n]) ? ");
                    gets(in_line);
                    if (strlen(in_line) > 0) {
                        for (i = 0; in_line[i] <= ' '; i++);
                        batch_overwrite = ((in_line[i] == 'y') || (in_line[i] == 'Y'));
                        }
                    }
                else
                    printf("File %s does not exist. Operating in interactive mode.\n",batch_file_name);
                }
            else
                printf("No file specified. Operating in interactive mode.\n");
            }
        }
}


/*****************************************************************************
   This routine returns one run code at a time to the mainline. If the program is run in batch mode, the
   run codes are extracted from the batch file, and the first byte in the batch file is checked to see if it is
   a normal format Word file. If the program is run in interactive mode, the user is prompted for the
   run code; AATM is terminated by entering a null run code (return value of 0). It is called from the
    mainline.
*****************************************************************************/
#define WORD_FLAG 0xFFFE

int get_run_code()
{
    int i, j, l;
    int retval = TRUE;

    if (batch_mode) {
        fgets(in_line,MAX_LINE_LENGTH-1,batch_fptr);
        if (in_line[0] == WORD_FLAG) {
           fseek(batch_fptr,(long)256,SEEK_SET);
           fgets(in_line,MAX_LINE_LENGTH-1,batch_fptr);
           }
        }
    else {
        printf("Enter run code:");
        gets(in_line);
        }

    l = strlen(in_line);
    i = 0;
    while (in_line[i] <= ' ') {    /* skip white space */
        i++;
        if (i >= l) break;
        }
    if (i < l) {       /* run code extends until first blank space or end of string */
        j = i;
        while (in_line[j] > ' ') {
            j++;
            if (j == l) break;
```

```
            }
        strncpy(run_code,&(in_line[i]),j-i);
        run_code[j-i] = '\0';
        }
    else {
        run_code[0] = '\0';
        retval = FALSE;   /* terminate AATM */
        }
    return(retval);
}


/*****************************************************************************
   This routine returns the calibration factors to the mainline. If the program is run in batch mode, the
   cal factors are extracted from the batch file;  if the program is run in interactive mode, the user is
   prompted for the cal factors. It is called from the mainline.
*****************************************************************************/
int get_cal_factors()
{
     double my_atof();

    if (batch_mode) {
        fgets(in_line,MAX_LINE_LENGTH-1,batch_fptr);
        hcal = my_atof(in_line);
        fgets(in_line,MAX_LINE_LENGTH-1,batch_fptr);
        vcal = my_atof(in_line);
        }
    else {
        printf("Enter horizontal calibration factor in deg/unit:");
        gets(in_line);
        hcal = my_atof(in_line);
        printf("Enter vertical calibration factor in deg/unit:");
        gets(in_line);
        vcal = my_atof(in_line);
        }
}



/*****************************************************************************
   This routine replaces the "atof" routine in the Think C library. The built-in routine ignores zeroes
   following the decimal place so that 0.05 and 0.5 are both interpreted as 0.5. This is disastrous
   for calibration factors. It is called from "get_cal_factors".
*****************************************************************************/
double my_atof(str)
char *str;
{
    double frac_scale = 1;
    double num = 0;
    int i, l;
    int neg_flag = FALSE;

    l = strlen(str);
    i = 0;

    while (str[i] <= ' ') {
        i++;
        if (i >= l) break;
```

```
        }
    if (i >= l)
        num = 0.0;
    else {
        if (str[i] == '-') {
            neg_flag = TRUE;
            i++;
            }
        else if (str[i] == '+') {
            i++;
            }
        while ((str[i] >= '0') && (str[i] <= '9')) {
            num *= 10;
            num += str[i] - '0';
            i++;
            }
        if (str[i] == '.') {
            i++;
            while ((str[i] >= '0') && (str[i] <= '9')) {
                frac_scale *= 0.1;
                num += (str[i] - '0') * frac_scale;
                i++;
                }
            }
        if (neg_flag)
            num = -num;
        }
    return(num);
}


/*****************************************************************************
   This routine opens the input eye position data file. If the file exists, the return value is 1;  if the file
   does not exist, an error message is generated, and the return value is 0.  It is called from the mainline.
*****************************************************************************/
int open_input_file(root_file)
char *root_file;
{
   void file_name();
    int l;
   char in_filename[MAX_LINE_LENGTH];
    int retval;

   file_name(run_code,root_file,in_filename);
   retval = open(in_filename,O_RDONLY|O_BINARY);
   if (retval <= 0) {

     retval = FALSE;
      printf("File %s does not exist.  Skipping this file.\n",in_filename);

      }
   return(retval);
}


/*****************************************************************************
   This routine creates an output data file for either filtered eye position, eye velocity, or SPV depending
```

```
   upon the "root_file" argument;  the root_file is one of the file name specifications (e.g. OSH_FILE).
   If the file already exists, and the program is in interactive mode, the user is prompted for overwrite
   permission.  If this is granted, the return value is 1;  otherwise, it is 0.  If the file already exists, and the
   program is in batch mode, the global "batch_overwrite" flag is checked.  it is called from the mainline.
*****************************************************************************/
int create_output_file(root_file)
char *root_file;
{
   void file_name();
    int l;
   char out_filename[MAX_LINE_LENGTH];
   int out_handle;
   FILE *out_fptr;
   int retval = TRUE;

   file_name(run_code,root_file,out_filename);
   out_handle = open(out_filename,O_BINARY|O_RDONLY);

   if (out_handle > 0) {
      if (batch_mode) {
         in_line[0] = (batch_overwrite) ? 'y' : 'n';
         }
      else {
         printf("Output file %s already exists.  Overwrite (y/n) ?", out_filename);
         gets(in_line);
         }
      if ((in_line[0] == 'y') || (in_line[0] == 'Y')) {
         printf("Overwriting %s.\n",out_filename);
         close(out_handle);
         remove(out_filename);
         out_fptr = fopen(out_filename,"wt");
         fclose(out_fptr);
         out_handle = open(out_filename,O_BINARY|O_RDWR|O_APPEND);
         retval = out_handle;
         }
      else {
         printf("Aborting for this file.\n");
         close(out_handle);
         retval = FALSE;
         }
      }
   else {
      printf("Creating new output file %s.\n",out_filename);
      out_fptr = fopen(out_filename,"wt");
      fclose(out_fptr);
      out_handle = open(out_filename,O_BINARY|O_RDWR|O_APPEND);
      retval = out_handle;
      }
   return(retval);
}


/*****************************************************************************
   This routine creates an output data file name "out_name" by replacing the # sign in "root_file" by
   the "run_code".  It is called from the "open_input_file" and "create_output_file" routines.
*****************************************************************************/
void file_name(run_code,root_file,out_name)
char *run_code, *root_file, *out_name;
```

```
{
   char *retval;
    int i, j, l1, l2;

   l1 = strlen(run_code);
   l2 = strlen(root_file);
   for (i = 0; i < l2; i++) {
      if (root_file[i] == '#')
         break;
       }
   if (i == l2)
      strcpy( out_name, root_file );
   else {
      strncpy( out_name, root_file, i );
      strncpy( &(out_name[i]), run_code, l1 );
      strncpy( &(out_name[i+l1]), &(root_file[i+1]), l2-i );
       }
}


/*****************************************************************************
   This routine reads in the header information for the raw eye position data file, and determines the
   input data type. The return value is 1 if the data file is OK, or 0 if the data type is unknown. It is
   called from the mainline.
*****************************************************************************/
int get_file_parameters(handle,err_name)
int handle;
char *err_name;
{
    int retval = TRUE;

   read(handle,(char *)(&header),sizeof(Fmatrix));
   num_samples = header.ncols * (size_t)header.mrows;
   if ((header.type == USI_ROW_TYPE) || (header.type == USI_COL_TYPE))
      input_data_type = SI;          /* sign doesn't matter */
   else if ((header.type == SI_ROW_TYPE) || (header.type == SI_COL_TYPE))
     input_data_type = SI;
   else if ((header.type == LI_ROW_TYPE) || (header.type == LI_COL_TYPE))
     input_data_type = LI;
   else if ((header.type == R4_ROW_TYPE) || (header.type == R4_COL_TYPE))
     input_data_type = R4;
   else if ((header.type == SD_ROW_TYPE) || (header.type == SD_COL_TYPE))
     input_data_type = SD;
    else {
      input_data_type = -1;
      printf("%s file is not proper Matlab file\n",err_name);
      retval = FALSE;
       }
   read(handle,matlab_name,header.namlen);
   return(retval);
}


/*****************************************************************************
   This routine outputs the header information for an output data file (MatLab format). It is called
   from "save_os_data", "save_vel_data", and from the mainline (for SPV data).
*****************************************************************************/
void write_matlab_header(handle,name)
```

```
int handle;
char *name;
{

   header.namlen = strlen(name) + 1;
   header.type = OUT_TYPE;
   write( handle, (char *)&header, sizeof(Fmatrix));
   write( handle, name, header.namlen);
}



/*****************************************************************************
   This routine loads the raw eye position data into the appropriate type of memory buffer, and then
   transfers it to the standard data type buffer, "pos_buffer". The first and last position data samples
   are repeated for an extra half second of padding at the beginning and end of the buffer; this gives
   the space required for AATM transients. It is called from the mainline.
*****************************************************************************/
void load_pos_data()
{
    int i, N;

    N = SAMPLE_RATE / 2;   /* one-half second */

    /* read in as many complete buffers as there are */
    for (i = 0; i < num_samples/NUM_BUFFER; i++) {
      switch (input_data_type) {
        case SI:
          READ_SI_BUFFER(pos_handle,&(si_buffer[i*NUM_BUFFER + N]),NUM_BUFFER)
          break;
        case LI:
          READ_LI_BUFFER(pos_handle,&(li_buffer[i*NUM_BUFFER + N]),NUM_BUFFER)
          break;
        case R4:
          READ_R4_BUFFER(pos_handle,&(r4_buffer[i*NUM_BUFFER + N]),NUM_BUFFER)
          break;
        case SD:
          READ_SD_BUFFER(pos_handle,&(sd_buffer[i*NUM_BUFFER + N]),NUM_BUFFER)
          break;
       }
      }


    /* read in remaining partial buffer */
    switch (input_data_type) {
      case SI:
        READ_SI_BUFFER(pos_handle, &(si_buffer[i*NUM_BUFFER + N]),
                                (num_samples%NUM_BUFFER))
        break;
      case LI:
        READ_LI_BUFFER(pos_handle, &(li_buffer[i*NUM_BUFFER + N]),
                                (num_samples%NUM_BUFFER))
        break;
      case R4:
        READ_R4_BUFFER(pos_handle, &(r4_buffer[i*NUM_BUFFER + N]),
                                (num_samples%NUM_BUFFER))
        break;
      case SD:
```

```
        READ_SD_BUFFER(pos_handle, &(sd_buffer[i*NUM_BUFFER + N]),
                                (num_samples%NUM_BUFFER))
        break;
      }

    /* transfer data to standardized short double pos_buffer */
    switch (input_data_type) {
      case SI:
        for (i = (num_samples + N - 1); i >= N; i--) {
          pos_buffer[i] = si_buffer[i];
          }
        break;
      case LI:
        for (i = (num_samples + N - 1); i >= N; i--) {
          pos_buffer[i] = li_buffer[i];
          }
        break;
      case R4:
        for (i = (num_samples + N - 1); i >= N; i--) {
          pos_buffer[i] = r4_buffer[i];
          }
        break;
      case SD:
        for (i = (num_samples + N - 1); i >= N; i--) {
          pos_buffer[i] = sd_buffer[i];
          }
        break;
      }

    for (i = 0; i < N; i++)
        pos_buffer[i] = pos_buffer[N];
    for (i = N; i < SAMPLE_RATE; i++)
        pos_buffer[num_samples+i] = pos_buffer[num_samples+N-1];

}


/*****************************************************************************
   This routine performs the PFMH filtering (two linear filters of lengths 6 and 10) on the eye position
   data. The weighting coefficients for a first-order (linear) are calculated in the PFMH1 routine. This
   routine only works as is for first order filters. The code has been optimized for execution speed by
   calculating some difference coefficients and maintaining running sums in x1F, x2F, x1B, and x2B;
   in particular, the difference between consecutive weights is assumed to be a constant, which is only
   true for zeroth- and first-order weights. For non-linear nystagmus patterns, such as those due to
   torsional and/or sinusoidal stimulation, at least one second-order filter should be used.

   This routine should be re-written in a more legible form and comparisons made to determine if there
   is a speed difference, now that the 68020 and 68881 math co-processor options are being used;
   before these options, this compact and illegible code was much faster. The "median5" call at the end
   should also be replaced by a more general sort routine for more than 5 estimates. Again, this was
   optimized for speed of execution.

   This routine is called from the mainline.
*****************************************************************************/
void OS_lin2()
{
```

```
#define N1 6
#define N2 10
    register size_t i, j;

    void PFMH1();
    double median5();
    size_t start, stop;
    double h1[N1], h2[N2];
    double x1F, x1B, x2F, x2B;

    PFMH1(N1,h1);
    PFMH1(N2,h2);

    start = SAMPLE_RATE / 2 - N2;
    stop = num_samples + SAMPLE_RATE / 2 + N2;
    for (i = start; i < stop; i++) {

        x1F = 0;
        x1B = 0;
        x2F = 0;
        x2B = 0;
        for (j = 0; j < N1; j++) {
            x1F += h1[j] * pos_buffer[i + 1 + j];
            x1B += h1[j] * pos_buffer[i - 1 - j];
            }
        for (j = 0; j < N2; j++) {
            x2F += h2[j] * pos_buffer[i + 1 + j];
            x2B += h2[j] * pos_buffer[i - 1 - j];
            }

        pos_buffer[i] = median5((double)pos_buffer[i],x1F,x1B,x2F,x2B);

        }
}



/*****************************************************************************
   This routine calculates the weighting coefficients for a zeroth-order filter. It is not currently used.
*****************************************************************************/
void PFMH0(N,h)
int N;
double h[];
{
    int i;

    for (i = 1; i <= N; i++)
        h[i-1] = 1/ (double)N;
}


/*****************************************************************************
   This routine calculates the weighting coefficients for a first-order filter. It is called from "OS_lin2".
*****************************************************************************/
void PFMH1(N,h)
int N;
double h[];
{
```

```
    int i;

    for (i = 1; i <= N; i++)
        h[i-1] = (4 * N - 6 * i + 2) / (double)(N * (N - 1));
}


/*****************************************************************************
  This routine calculates the weighting coefficients for a second-order filter.  It is not currently used.
*****************************************************************************/
void PFMH2(N,h)
int N;
double h[];
{
    int i;

    for (i = 1; i <= N; i++)
        h[i] = (9 * N * N + (9 - 36 * i) * N + 30 * i * i - 18 * i + 6) /  (double)(N * (N * N - 3 * N +2));
}



/*****************************************************************************
  This routine calculates the eye velocity by scaling the position data by the appropriate calibration
  factor, and differentiating the result (digital filter).  it is called from the mainline.
*****************************************************************************/
void calc_vel_data(cal_factor)
double cal_factor;
{
    int N;
    size_t i, j, k;
     double scale;

    N = SAMPLE_RATE / 2;
    scale = cal_factor / velA;

    /* differentiate eye position to get eye velocity */
    i = N;
    for (k = 0; k < (N - (FILTER_LENGTH / 2)); k++)
        vel_buffer[k] = 0;
    for ( ; k < (num_samples + N + (FILTER_LENGTH / 2)); k++) {
        vel_buffer[k] = 0;
        for (j = 0; j < FILTER_LENGTH; j++) {
            vel_buffer[k] += velB[j] * pos_buffer[i-j];
            }
        vel_buffer[k] *= scale;
        i++;
        }
    for ( ; k < num_samples + SAMPLE_RATE; k++)
        vel_buffer[k] = 0;
}


/*****************************************************************************
  This routine saves the velocity data to disk.  It is called from the mainline.
*****************************************************************************/
void save_vel_data(var_name)
char *var_name;
```

```
{
    int i, N;

    write_matlab_header(vel_handle,var_name);

    N = SAMPLE_RATE / 2;
    for (i = 0; i < num_samples/NUM_BUFFER; i++)
        WRITE_SD_BUFFER(vel_handle,&(vel_buffer[i*NUM_BUFFER + N]),NUM_BUFFER)

    WRITE_SD_BUFFER(vel_handle, &(vel_buffer[i*NUM_BUFFER+N]),
                        (num_samples%NUM_BUFFER))

}


/*****************************************************************************
  This routine saves the PFMH-filtered eye position data to disk. It is called from the mainline.
*****************************************************************************/
void save_os_data(var_name)
char *var_name;
{
    int i, N;

   write_matlab_header(os_handle,var_name);

    N = SAMPLE_RATE / 2;
    for (i = 0; i < num_samples/NUM_BUFFER; i++)
       WRITE_SD_BUFFER(os_handle,&(pos_buffer[i*NUM_BUFFER + N]),NUM_BUFFER)

   WRITE_SD_BUFFER(os_handle, &(pos_buffer[i*NUM_BUFFER+N]),
           (num_samples%NUM_BUFFER))

}


/*****************************************************************************
  This is the central AATM routine which estimates the slow phase velocity (SPV). Parameters are
  initialized for the algorithm. The first second of data is sorted by a bubble sort (order N^2), which
  will suffice since it is only performed once and there are only 121 points (SAMPLE_RATE+1)
  being sorted. The skewness of the first second of data is calculated, and then the SPV at the centre
  of the window (actually at t=0 since the data was padded by a half-second) is calculated.

  For each subsequent sample, the one-second window is shifted right by one data point so that a
  "new" value appears in the window and an "old" value has passed out of the window. The sorted
  list is binary-searched (order logN) for the old value and the insertion point for the new value. Then,
  the old value is deleted and the new value added. This operation is order 3logN which is much better
  than sorting the entire second of data (order NlogN even for a quicksort).

  For each sample, the skewness and SPV are calculated. When the index pointer passes a multiple
  of 4000 samples (I/O buffer size), the processed portion of the SPV is written to the disk. At the
  end of the routine, the data files are all closed.

  This routine is called from the mainline.
*****************************************************************************/
void AATM()
{
```

```
void bubble_sort();
int check_sort();
int i, j, k, k1, k2;
int n, n_out , in_bytes;
int N, stop, L, K;
int Lalpha, Lbeta, M;
short double s[SAMPLE_RATE + 1];
double Sbeta, sum;
short double old, new;

printf("Total number of samples = %ld\n",num_samples);
if (num_samples <= SAMPLE_RATE) {
    printf("Input file is too short.\n");
    return;
    }

/* initialize parameters, using a one-second window */
N = SAMPLE_RATE / 2;
L = SAMPLE_RATE + 1;

Lalpha = L * ALPHA;
Lbeta = L * BETA;
M = L * MU;


/* initialize and sort first second of data */
memcpy( (void *)s, (void *)vel_buffer, (size_t)(L * sizeof(short double)) );
bubble_sort(s,L);

/* calculate skewness of first second of data */
if (s[L-Lbeta-1] == s[Lbeta-1])
    Sbeta = 0;
else
    Sbeta = (s[L-Lbeta-1] + s[Lbeta-1] - 2*s[N]) / (s[L-Lbeta-1] - s[Lbeta-1]);
K = - Sbeta * M;

/* new value is mean of estimated peak of histogram */
sum = 0;
for (i = (Lalpha+K); i < (L-Lalpha+K); i++)
    sum = sum + s[i];
spv_buffer[N] = sum / (L - 2*Lalpha);

stop = num_samples + N;
n = N + 1;

while (in_samples > 0) {   /* more data to sort */

    while (n < stop) {     /* continue processing this buffer */

        n_out = n % NUM_BUFFER;   /* index within output buffer */

        /* check for full output buffer */
        if (n_out == 0) {
            if (n == NUM_BUFFER)
                WRITE_SD_BUFFER(spv_handle,&(spv_buffer[N]),NUM_BUFFER-N)
            else
                WRITE_SD_BUFFER(spv_handle,spv_buffer,NUM_BUFFER)
            }
```

```
old = vel_buffer[n - N - 1];  /* value to be removed from sorted list */
new = vel_buffer[n + N];    /* value to be inserted into sorted list */

/* binary search for old value */
i = 0;
j = L-1;
k1 = (i+j) / 2;
while (i != k1) {        /* s(i) <= old <= s(i+1) */
    if (s[k1] == old)
        break;
    else if (s[k1] < old)
        i = k1;
    else
        j = k1;
    k1 = (i+j) / 2;
    }
if (s[k1+1] == old)
    k1 = k1 + 1;
/* we now have s[k1] = old */

/* check for new value out of range of sorted list */
if (new < s[0]) {   /* insert at beginning of list */
    if (k1 == 0)
        s[0] = new;
    else {
        for (k = k1; k > 0; k--)
            s[k] = s[k-1];
        s[0] = new;
        }
    }
else if (new > s[L-1]) {   /* insert at end of list */
    if (k1 == (L-1))
        s[L-1] = new;
    else {
        for (k = k1; k < (L-1); k++)
            s[k] = s[k+1];
        s[L-1] = new;
        }
    }
else {   /* binary search for location to insert new value */

    /* reset i,j to optimize search for new value */
    if (new > old) {
        i = k1;
        j = L-1;
        }
    else if (new < old) {
        i = 0;
        j = k1;
        }
    else {
        i = k1;
        j = k1;
        }

    k2 = (i+j) / 2;
    while (i != k2) {        /* s(i) <= new <= s(i+1) */
```

```
                    if (s[k2] == new)
                        break;
                    else if (s[k2] < new)
                        i = k2;
                    else
                        j = k2;
                    k2 = (i+j) / 2;
                    }
                /* we now have s[k2] <= new <= s[k2+1] */

                if (k1 == k2)
                    s[k1] = new;
                else if (k1 < k2) {
                    for (k = k1; k < k2; k++)
                        s[k] = s[k+1];
                    s[k2] = new;
                    }
                else {
                    for (k = k1; k > k2; k--)
                        s[k] = s[k-1];
                    s[k2+1] = new;
                    }

                }


            /* calculate skewness of histogram */
            if (s[L-Lbeta-1] == s[Lbeta-1])
                  Sbeta = 0;
            else
                 Sbeta = (s[L-Lbeta-1] + s[Lbeta-1] - 2*s[N]) / (s[L-Lbeta-1] - s[Lbeta-1]);
            K = - Sbeta * M;

            /* new value is mean of estimated peak of histogram */
            sum = 0;
            for (i = (Lalpha+K); i < (L-Lalpha+K); i++)
                sum = sum + s[i];
            spv_buffer[n_out] = sum / (L - 2*Lalpha);

            n++;
            }

        WRITE_SD_BUFFER(spv_handle,spv_buffer,n_out+1)
        break;

        }

    close(pos_handle);
    close(vel_handle);
    close(spv_handle);
}



/*****************************************************************************
  This routine performs a standard bubble sort algorithm on the data in "x".  It is called from "AATM".
*****************************************************************************/
```

```
void bubble_sort(x,n)
short double *x;
int n;
{
    int i, j;
    short double mx;
    int imx;

    for (i = n-1; i > 0; i--) {
        imx = i;
        mx = x[imx];
        for (j = i - 1; j >= 0; j--) {
            if (x[j] > mx) {
                imx = j;
                mx = x[imx];
                }
            }
        x[imx] = x[i];
        x[i] = mx;
        }
}


/*****************************************************************************
   This routine calculates the median of the five arguments. The various comments show the order of
   the five values, in descending order, as deduced to that point. Once the order of the five points has
   been calculated, the median is simply the middle point. It is called from "OS_lin2".
*****************************************************************************/
double median5(x1,x2,x3,x4,x5)
double x1, x2, x3, x4, x5;
{
    double m;

    if (x1 >= x2) {    /* 1,2 */
        if (x2 >= x3) {    /* 1,2,3 */
            if (x3 >= x4) {    /* 1,2,3,4 */
                if (x3 >= x5) {
                    m = x3;       /* 1,2,3,4,5 or 1,2,3,5,4 */
                    }
                else if (x2 >= x5) {
                    m = x5;       /* 1,2,5,3,4 */
                    }
                else {
                    m = x2;       /* 1,5,2,3,4 or 5,1,2,3,4 */
                    }
                }
            else if (x2 >= x4) {    /* 1,2,4,3 */
                if (x4 >= x5) {
                    m = x4;       /* 1,2,4,5,3 or 1,2,4,3,5 */
                    }
                else if (x2 >= x5) {
                    m = x5;       /* 1,2,5,4,3 */
                    }
                else {
                    m = x2;       /* 1,5,2,4,3 or 5,1,2,4,3 */
                    }
                }
```

```
    else if (x1 >= x4) {    /* 1,4,2,3 */
        if (x2 >= x5) {
            m = x2;         /* 1,4,2,3,5 or 1,4,2,5,3 */
            }
        else if (x4 >= x5) {
            m = x5;         /* 1,4,5,2,3 */
            }
        else {
            m = x4;         /* 1,5,4,2,3 or 5,1,4,2,3 */
            }
        }
    else {                  /* 4,1,2,3 */
        if (x2 >= x5) {
            m = x2;         /* 4,1,2,3,5 or 4,1,2,5,3 */
            }
        else if (x1 >= x5) {
            m = x5;         /* 4,1,5,2,3 */
            }
        else {
            m = x1;         /* 4,5,1,2,3 or 5,4,1,2,3 */
            }
        }
    }
else if (x1 >= x3) {    /* 1,3,2 */
    if (x2 >= x4) {     /* 1,3,2,4 */
        if (x2 >= x5) {
            m = x2;         /* 1,3,2,4,5 or 1,3,2,5,4 */
            }
        else if (x3 >= x5) {
            m = x5;         /* 1,3,5,2,4 */
            }
        else {
            m = x3;         /* 1,5,3,2,4 or 5,1,3,2,4 */
            }
        }
    else if (x3 >= x4) {    /* 1,3,4,2 */
        if (x4 >= x5) {
            m = x4;         /* 1,3,4,5,2 or 1,3,4,2,5 */
            }
        else if (x3 >= x5) {
            m = x5;         /* 1,3,5,4,2 */
            }
        else {
            m = x3;         /* 1,5,3,4,2 or 5,1,3,4,2 */
            }
        }
    else if (x1 >= x4) {    /* 1,4,3,2 */
        if (x3 >= x5) {
            m = x3;         /* 1,4,3,2,5 or 1,4,3,5,2 */
            }
        else if (x4 >= x5) {
            m = x5;         /* 1,4,5,3,2 */
            }
        else {
            m = x4;         /* 1,5,4,3,2 or 5,1,4,3,2 */
            }
        }
    else {                  /* 4,1,3,2 */
```

```
            if (x3 >= x5) {
                m = x3;       /* 4,1,3,2,5 or 4,1,3,5,2 */
                }
            else if (x1 >= x5) {
                m = x5;       /* 4,1,5,3,2 */
                }
            else {
                m = x1;       /* 4,5,1,3,2 or 5,4,1,3,2 */
                }
            }
        }
    else {              /* 3,1,2 */
        if (x2 >= x4) {     /* 3,1,2,4 */
            if (x2 >= x5) {
                m = x2;       /* 3,1,2,4,5 or 3,1,2,5,4 */
                }
            else if (x1 >= x5) {
                m = x5;       /* 3,1,5,2,4 */
                }
            else {
                m = x1;       /* 3,5,1,2,4 or 5,3,1,2,4 */
                }
            }
        else if (x1 >= x4) {    /* 3,1,4,2 */
            if (x4 >= x5) {
                m = x4;       /* 3,1,4,5,2 or 3,1,4,2,5 */
                }
            else if (x1 >= x5) {
                m = x5;       /* 3,1,5,4,2 */
                }
            else {
                m = x1;       /* 3,5,1,4,2 or 5,3,1,4,2 */
                }
            }
        else if (x3 >= x4) {    /* 3,4,1,2 */
            if (x1 >= x5) {
                m = x1;       /* 3,4,1,2,5 or 3,4,1,5,2 */
                }
            else if (x4 >= x5) {
                m = x5;       /* 3,4,5,1,2 */
                }
            else {
                m = x4;       /* 3,5,4,1,2 or 5,3,4,1,2 */
                }
            }
        else {              /* 4,3,1,2 */
            if (x1 >= x5) {
                m = x1;       /* 4,3,1,2,5 or 4,3,1,5,2 */
                }
            else if (x3 >= x5) {
                m = x5;       /* 4,3,5,1,2 */
                }
            else {
                m = x3;       /* 4,5,3,1,2 or 5,4,3,1,2 */
                }
            }
        }
}
```

```
else {              /* 2,1 */
    if (x1 >= x3) {    /* 2,1,3 */
        if (x3 >= x4) {    /* 2,1,3,4 */
            if (x3 >= x5) {
                m = x3;       /* 2,1,3,4,5 or 2,1,3,5,4 */
                }
            else if (x1 >= x5) {
                m = x5;       /* 2,1,5,3,4 */
                }
            else {
                m = x1;       /* 2,5,1,3,4 or 5,2,1,3,4 */
                }
            }
        else if (x1 >= x4) {    /* 2,1,4,3 */
            if (x4 >= x5) {
                m = x4;       /* 2,1,4,5,3 or 2,1,4,3,5 */
                }
            else if (x1 >= x5) {
                m = x5;       /* 2,1,5,4,3 */
                }
            else {
                m = x1;       /* 2,5,1,4,3 or 5,2,1,4,3 */
                }
            }
        else if (x2 >= x4) {    /* 2,4,1,3 */
            if (x1 >= x5) {
                m = x1;       /* 2,4,1,3,5 or 2,4,1,5,3 */
                }
            else if (x4 >= x5) {
                m = x5;       /* 2,4,5,1,3 */
                }
            else {
                m = x4;       /* 2,5,4,1,3 or 5,2,4,1,3 */
                }
            }
        else {              /* 4,2,1,3 */
            if (x1 >= x5) {
                m = x1;       /* 4,2,1,3,5 or 4,2,1,5,3 */
                }
            else if (x2 >= x5) {
                m = x5;       /* 4,2,5,1,3 */
                }
            else {
                m = x2;       /* 4,5,2,1,3 or 5,4,2,1,3 */
                }
            }
        }
    else if (x2 >= x3) {    /* 2,3,1 */
        if (x1 >= x4) {    /* 2,3,1,4 */
            if (x1 >= x5) {
                m = x1;       /* 2,3,1,4,5 or 2,3,1,5,4 */
                }
            else if (x3 >= x5) {
                m = x5;       /* 2,3,5,1,4 */
                }
            else {
                m = x3;       /* 2,5,3,1,4 or 5,2,3,1,4 */
                }
```

```
        }
    else if (x3 >= x4) {    /* 2,3,4,1 */
        if (x4 >= x5) {
            m = x4;         /* 2,3,4,5,1 or 2,3,4,1,5 */
            }
        else if (x3 >= x5) {
            m = x5;         /* 2,3,5,4,1 */
            }
        else {
            m = x3;         /* 2,5,3,4,1 or 5,2,3,4,1 */
            }
        }
    else if (x2 >= x4) {    /* 2,4,3,1 */
        if (x3 >= x5) {
            m = x3;         /* 2,4,3,1,5 or 2,4,3,5,1 */
            }
        else if (x4 >= x5) {
            m = x5;         /* 2,4,5,3,1 */
            }
        else {
            m = x4;         /* 2,5,4,3,1 or 5,2,4,3,1 */
            }
        }
    else {              /* 4,2,3,1 */
        if (x3 >= x5) {
            m = x3;         /* 4,2,3,1,5 or 4,2,3,5,1 */
            }
        else if (x2 >= x5) {
            m = x5;         /* 4,2,5,3,1 */
            }
        else {
            m = x2;         /* 4,5,2,3,1 or 5,4,2,3,1 */
            }
        }
    }
else {              /* 3,2,1 */
    if (x1 >= x4) {     /* 3,2,1,4 */
        if (x1 >= x5) {
            m = x1;         /* 3,2,1,4,5 or 3,2,1,5,4 */
            }
        else if (x2 >= x5) {
            m = x5;         /* 3,2,5,1,4 */
            }
        else {
            m = x2;         /* 3,5,2,1,4 or 5,3,2,1,4 */
            }
        }
    else if (x2 >= x4) {    /* 3,2,4,1 */
        if (x4 >= x5) {
            m = x4;         /* 3,2,4,5,1 or 3,2,4,1,5 */
            }
        else if (x2 >= x5) {
            m = x5;         /* 3,2,5,4,1 */
            }
        else {
            m = x2;         /* 3,5,2,4,1 or 5,3,2,4,1 */
            }
        }
```

```
            else if (x3 >= x4) {    /* 3,4,2,1 */
                if (x2 >= x5) {
                    m = x2;         /* 3,4,2,1,5 or 3,4,2,5,1 */
                    }
                else if (x4 >= x5) {
                    m = x5;         /* 3,4,5,2,1 */
                    }
                else {
                    m = x4;         /* 3,5,4,2,1 or 5,3,4,2,1 */
                    }
                }
            else {              /* 4,3,2,1 */
                if (x2 >= x5) {
                    m = x2;         /* 4,3,2,1,5 or 4,3,2,5,1 */
                    }
                else if (x3 >= x5) {
                    m = x5;         /* 4,3,5,2,1 */
                    }
                else {
                    m = x3;         /* 4,5,3,2,1 or 5,4,3,2,1 */
                    }
                }
            }
        }
    return(m);
}


/*****************************************************************************
   This routine ensures that the data in "x" is sorted in ascending order.  It is for debugging purposes.
*****************************************************************************/
int check_sort(x,n)
short double *x;
int n;
{
    int i;
    int retval = 0;

    for (i = n - 1; i > 0; i--) {
        if (x[i] < x[i-1]) {
            retval = TRUE;
            break;
            }
        }
    return(retval);
}
```

Appendix G

Tach and Button Push Analysis Software

tachan

delta_tach

collect_parms

find_entry

T_check

mean_butt_data

output_stats

```
%tachan
%
% This analyzes a tach signal to determine the start of the chair
% acceleration, the spin length, the spin velocity, the times of
% the per- and post-rotatory button pushes, and the length of the
% overall run. The basic concept is that a chair acceleration
% or deceleration, or a button push, is mirrored by a large
% change in the tach signal; and that button pushes and chair
% start/stop have different profiles. A button push is a sudden
% change of 0.87 V; whereas, a chair start or stop is an
% exponential change in the signal with a time constant of
% 0.17 seconds. Thresholds can differentiate between the two.
% The six parameters are saved in a .parms file.
%
% Written by D. Balkwill and C. Pouliot
% Last Modification: 12/22/91

%load tach data into standardized variable
run_code = input('Enter Run Code: ','s');
file_specs
eval(['load ',data_path,file_name(Tach_File,run_code)]);
eval(['TACH = ',Tach_Var,';']);
eval(['clear ',Tach_Var]);

%convert to deg/sec
TACH = TACH / 204.8;        % 10 V == 2048 units
TACH = TACH * 120 / 2.08;   % 120 deg/sec = 2.08 V

%get sample rate
nysa_path = get_path;
eval(['load ',nysa_path,'bookkeeping:vel_filter.mat']);
clear A B

l = length(TACH);
t = [0:l-1] / sample;

steady_vel = 120;  %ideal chair spin rate
but_thresh = 40;   %threshold above which a button push must be


%look for events (large changes) until start of motion is found
event = delta_tach(TACH,1);
while (abs(TACH(event + sample)) < steady_vel/2)
   %if start, then chair must be up to speed one second later
   event = delta_tach(TACH,event+1);
end
start = event;

% look for next event, and calculate average velocity as
% mean value between (start + 3 seconds) and this event
event = delta_tach(TACH,start + 3 * sample);  %T1 > 3 sec
avg = mean(TACH(start + 3 * sample:event - 5));
level = mean(TACH(event + 10:event + 20));
```

```
% if level too low for button push, keep scanning
while (abs(level - avg) < but_thresh)  % noise artifact
   event = delta_tach(TACH,event+1);
   level = mean(TACH(event + 10:event + 20));
end

if ((mean(TACH(event+3:event+8)) - avg) < but_thresh)  %not a button push
   stop = event;
   but1 = NaN;
else
   but1 = event;
   level = but_thresh + avg;  %threshold level

   % look for end of button push
   event = event + 10;
   while (mean(TACH(event:event+2)) > level)
      event = event + 10;
   end

   % look for chair stop, skipping button pushes if needed
   event = delta_tach(TACH,event+10);
   while (abs(TACH(event + sample)) > but_thresh)   % button push or noise
      event = event + 10;
      while (TACH(event) > level)   % skip to end of button push
         event = event + 10;
      end
      event = delta_tach(TACH,event);
   end
   stop = event;

end

% assume button push is at least 1.5 seconds after stop
flag = 1;
event = delta_tach(TACH,stop + 1.5 * sample);
if (event == NaN)   % no button push
   flag = 0;
elseif (TACH(event + 5) > but_thresh)  % noise artifact
   flag = 0;
end
while (flag)
   event = delta_tach(TACH,event+1);
   if (event == NaN)   % no button push
      flag = 0;
   elseif (TACH(event + 5) > but_thresh)  % noise artifact
      flag = 0;
   end
end
but2 = event;

spinv = avg;
delay = (start - 1) / sample;
T1 = (but1 - start) / sample;
T2 = (but2 - stop) / sample;
```

```
spinl = (stop - start) / sample;
runlen = t(l);

fprintf('Delay = %6.2f seconds.\n',delay);
fprintf('Spin length = %6.2f seconds.\n',spinl);
fprintf('Spin velocity = %6.2f deg/sec.\n',spinv);
fprintf('Per-rot button = %6.2f seconds.\n',T1);
fprintf('Post-rot button = %6.2f seconds.\n',T2);
fprintf('Run length = %6.2f seconds.\n',runlen);

clear event start stop but1 but2 sample avg but_thresh steady_vel flag level

eval(['save ',data_path,file_name(Parms_File,run_code),' spinv delay T1 T2 spinl
runlen']);

hold off
plot(t,TACH)
hold on
plot([delay,delay],[-500,500],'w')
plot([T1+delay,T1+delay],[-500,500],'b')
plot([spinl+delay,spinl+delay],[-500,500],'w')
plot([T2+spinl+delay,T2+spinl+delay],[-500,500],'b')
plot([delay,delay+spinl],[spinv,spinv],'g')
hold off


clear spinv delay T1 T2 spinl runlen t l TACH run_code data_path nysa_path
clear_specs
```

```
function n = delta_tach(tach,init)

%delta_tach
%
% This function looks for a large change in the tach signal
% (greater than 6 times the RMS difference, Chauvenet criterion),
% starting at a given index "init". The RMS is based upon the
% first 100 samples. The code is illegible at times since it
% is speed-optimized.
%
% Written by D. Balkwill
% Last Modification: 12/22/91

x = abs(tach);  %large increases and decreases are both detected
l = length(x);
index = [1:l]';
index(1:init) = 1e10 * ones(1,init);  %points already processed
change = 0;

false  = 0;
true = 1;

final = init + 100;

%set up 6 RMS bandwidth
themean = mean(x(init:final));
thestd = 6*std(x(init:final));
thresholdup = themean + thestd;
thresholddown = themean - thestd;

i = init;
condition = false;
while (condition == false)

                                if (i >= (l - 5))   % end of trace -- no large changes
                                   condition = true;
  else

     % y = 1e10, if point is within 6 RMS band
     % y = sample number, if outside 6 RMS band
     % therefore, min(y) is first point outside 6 RMS band
     y = ((x >= thresholdup) + (x <= thresholddown)) .* index;
     y = y + ((y == 0) * 1e10);
     change = min(y);

     if (change >= 1e10)  % no points outside 6 RMS, stop now
        condition = true;
        break;
     end

     % look for three consecutive points outside 6 RMS band,
     %   to minimize effects of spike noise artifacts
                                if (x(change) >= thresholdup)
        if ((x(change+1) >= thresholdup) & (x(change+2) >= thresholdup))
```

```
                                                    condition = true;
        end
                                elseif (x(change) <= thresholddown)
        if ((x(change+1) <= thresholddown) & (x(change+2) <= thresholddown))
                                                        condition = true;
        end
      end

      index(i+1:change) = 1e10 * ones(1,change-i);  % just processed
      i = change;

    end
end

n = i - 1;  % back up one point, for transients
if ((n >= (1 - 6)) | (change >= 1e10))   % no change
          n = NaN;
end

fprintf('End of delta_tach.\n');
return;
```

```
% collect_parms
%
% Collects all the run parameters into a single matrix.  Each
% row contains the data for one run.  The columns are as follows:
%     column#1:  Subject number (M = 1, N = 2, P = 3, T = 4)
%     column#2:  BDC session number (between 2 and 10)
%     column#3:  run number (2, 3, 4, 5, 7, 8, 9, or 10)
%     column#4:  T1 (per-rotatory button push time)
%     column#5:  T2 (post-rotatory button push time)
%     column#6:  delay (time delay before start of run)
%     column#7:  spinl (time length of spin portion)
%     column#8:  runlen (time length of un-normalized data file)
%     column#9:  spinv (mean spin velocity)
%     column#10:  gain1 (per-rot gain from exponential fit)
%     column#11:  gain2 (post-rot gain from exponential fit)
%     column#12:  tau1 (per-rot time constant from exponential fit)
%     column#13:  tau2 (post-rot time constant from exponential fit)
% Rows are not saved in the matrix for runs for which no data
% exists (i.e. runs were not performed, or data was lost).  The
% matrix is saved as ASCII data in the file "all_parms".
%
% A corresponding list of the run codes is saved in the file
% "all_codes" with the same variable name.  Since the run codes
% can be either 4 or 5 characters long, a blank character is
% added to the end of the 4-character run codes.
%
% D. Balkwill  9/19/91

file_specs

subject = ['M', 'N', 'P', 'T'];
session = [2, 3, 4, 5, 6, 7, 8, 9, 10];
runs = ['02'; '03'; '04'; '05'; '07'; '08'; '09'; '10'];

all_codes = [];
subj_vector = [];
BDC_vector = [];
run_vector = [];
T1_vector = [];
T2_vector = [];
delay_vector = [];
spinl_vector = [];
runlen_vector = [];
spinv_vector = [];
gain1_vector = [];
gain2_vector = [];
tau1_vector = [];
tau2_vector = [];

for i=1:max(size(subject))
 for j=1:max(size(session))
  for k=1:max(size(runs))
   run_code = [subject(i), int2str(session(j)), runs(k,:)];
   fname = [data_path,subject(i),':',file_name(Parms_File,run_code)];
```

```
    if (exist(fname) == 2)  % parms file exists
      eval(['load ',fname]);
      if (length(run_code) == 4)
        run_code = [run_code,' '];
      end
      all_codes = [all_codes; run_code];
      subj_vector = [subj_vector; i];
      BDC_vector = [BDC_vector; session(j)];
      run_vector = [run_vector; eval(runs(k,:))];
      T1_vector = [T1_vector; T1];
      T2_vector = [T2_vector; T2];
      delay_vector = [delay_vector; delay];
      spinl_vector = [spinl_vector; spinl];
      runlen_vector = [runlen_vector; runlen];
      spinv_vector = [spinv_vector; spinv];
      gain1_vector = [gain1_vector; gain1];
      gain2_vector = [gain2_vector; gain2];
      tau1_vector = [tau1_vector; tau1];
      tau2_vector = [tau2_vector; tau2];
    end
  end
 end
end

all_parms = [subj_vector, BDC_vector, run_vector, T1_vector, T2_vector, delay_vector,
spinl_vector, runlen_vector, spinv_vector, gain1_vector, gain2_vector, tau1_vector,
tau2_vector];

eval(['save ',data_path,'all_parms all_parms /ascii /single']);
eval(['save ',data_path,'all_codes all_codes']);

clear_specs
clear all_parms all_codes data_path
clear i j k subject session runs
clear run_code fname T1 T2 delay spinl runlen spinv
clear gain1 gain2 tau1 tau2
clear subj_vector BDC_vector run_vector T1_vector T2_vector
clear delay_vector spinl_vector runlen_vector spinv_vector
clear gain1_vector gain2_vector tau1_vector tau2_vector
```

```
function index = find_entry(hash_list,run_code)

% find_entry
%
% Calculates a hash value for the specified run code, and
% performs a binary search for its entry in the hash list.
% The row index is returned if the entry exists, and 0 is
% returned if it does not. The subjects are assigned numbers
% corresponding to that in the "all_parms" file: M=1, N=2,
% P=3, T=4. The BDC number is between 2 and 10. The run
% number is between 2 and 10. This hashing technique is
% much faster than a search for the run code itself.
%
% D. Balkwill 9/20/91

%
% determine hash value for run_code
% hash formula:   x = 400 * subject number + 20 * BDC + run
%
if (length(run_code) == 4)
   check = (20 * eval(run_code(2))) + eval(run_code(3:4));
else
   check = (20 * eval(run_code(2:3))) + eval(run_code(4:5));
end
if (run_code(1) == 'M')
   check = check + 400;
elseif (run_code(1) == 'N')
   check = check + 800;
elseif (run_code(1) == 'P')
   check = check + 1200;
else
   check = check + 1600;
end

% simple binary search for check value within sorted hash_list

a = 1;
b = length(hash_list);
c = floor((a+b)/2);
index = 0;
while (b > (a+1))
   if (hash_list(c) == check)
      index = c;
      break;
   elseif (hash_list(c) < check)
      a = c;
   else
      b = c;
   end
   c = floor((a+b)/2);
end

if (index == 0)
   if (hash_list(a) == check)
```

```
      index = a;
   elseif (hash_list(b) == check)
      index = b;
   end
end

return;
```

```
% T_check
%
% Allows the user to check the button push times that are stored
% in the "all_parms" file.  A pseudo-hash formula is used so that
% each run code corresponds to a unique number, and the hashed
% entries are in monotonically increasing order.  The user then
% supplies run codes, one at a time, corresponding to the runs to
% be checked.  The run code is hased, and then searched for by a
% binary search in the "find_entry" function.  If the run has no
% entry (not performed, or lost data), this is so indicated.
% Otherwise, the button push times T1 and T2 are displayed and
% can be changed by the user.  If the user does not want to
% change one of the times, specifying no time (i.e. hitting
% return will preserve the old time.  The program ends when the
% user specifies a null run code.
%
% D. Balkwill  9/20/90

file_specs
clear_specs

eval(['load ',data_path,'all_parms']);

%
% hash formula:    x = 400 * subject number + 20 * BDC + run
%
hash_list = 400 * all_parms(:,1) + 20 * all_parms(:,2) + all_parms(:,3);

run_code = input('Enter Run Code: ','s');
while (length(run_code))
   index = find_entry(hash_list,run_code);
   if (index)
       fprintf(['Entry for ',run_code,' at index ',int2str(index),'.\n']);
      fprintf('T1 = %6.2f seconds.\n',all_parms(index,4));
      fprintf('T2 = %6.2f seconds.\n',all_parms(index,5));
      yn = input('Are these times OK ([y]/n)? ','s');
     if (isempty(yn))
         yn = 'y';
     end
      if ((yn(1) == 'n') | (yn(1) == 'N'))
        T1 = input('New T1 value: ');
        if (~isempty(T1))
           all_parms(index,4) = T1;
        end
        T2 = input('New T2 value: ');
        if (~isempty(T2))
           all_parms(index,5) = T2;
        end
     end
   else
       fprintf(['No entry for ',run_code,'.\n']);
   end
   run_code = input('Enter Run Code: ','s');
end
```

```
eval(['save ',data_path,'all_parms all_parms']);

clear data_path all_parms hash_list
clear run_code T1 T2 index yn
```

```
% mean_butt_data
%
% This script calculates means and variances for the T1 and T2
% button push times for each subject.  Averages are across BDCs
% within stimulus types, and across stimulus types within BDCs.
%
% D. Balkwill 9/26/91

if (exist('hash_list') ~= 1)
   file_specs
   clear_specs
    eval(['load ',data_path,'all_parms']);
   hash_list = 400 * all_parms(:,1) + 20 * all_parms(:,2) + all_parms(:,3);
end

subject = ['M', 'N', 'P', 'T'];
session = [2, 3, 4, 5];
runs = ['02'; '03'; '04'; '05'; '07'; '08'; '09'; '10'];
CW_flag = [1, 0, 1, 0, 1, 0, 1, 0];
dump_flag = [0, 0, 1 ,1, 0, 0, 1, 1];

for i=1:max(size(subject))   % for each subject

  fprintf(['\n\nSubject ',subject(i),':\n==========\n\n']);

  fprintf('Average data across BDC2-5:\n');

  CW_hup_T1 = [];
  CW_hup_T2 = [];
  CCW_hup_T1 = [];
  CCW_hup_T2 = [];
  CW_dump_T1 = [];
  CW_dump_T2 = [];
  CCW_dump_T1 = [];
  CCW_dump_T2 = [];

  for j=1:max(size(runs))
    T1 = [];
    T2 = [];
    for k=1:max(size(session))
      run_code = [subject(i), int2str(session(k)), runs(j,:)];
      index = find_entry(hash_list,run_code);
      if (index > 0)
        if (all_parms(index,4) ~= NaN)
          T1 = [T1; all_parms(index,4)];
        end
        if (all_parms(index,5) ~= NaN)
          T2 = [T2; all_parms(index,5)];
        end
      end
    end

     output_stats(['Run# ',runs(j,:)],T1,T2);
```

```
  if (CW_flag(j))
   if (dump_flag(j))
    CW_dump_T1 = [CW_dump_T1; T1];
    CW_dump_T2 = [CW_dump_T2; T2];
   else
    CW_hup_T1 = [CW_hup_T1; T1];
    CW_hup_T2 = [CW_hup_T2; T2];
   end
  else
   if (dump_flag(j))
    CCW_dump_T1 = [CCW_dump_T1; T1];
    CCW_dump_T2 = [CCW_dump_T2; T2];
   else
    CCW_hup_T1 = [CCW_hup_T1; T1];
    CCW_hup_T2 = [CCW_hup_T2; T2];
   end
  end

 end

 output_stats('CW h-up',CW_hup_T1,CW_hup_T2);
 output_stats('CCW h-up',CCW_hup_T1,CCW_hup_T2);
 output_stats('CW dump',CW_dump_T1,CW_dump_T2);
 output_stats('CCW dump',CCW_dump_T1,CCW_dump_T2);
 output_stats('CW',[CW_hup_T1; CW_dump_T1],[CW_hup_T2; CW_dump_T2]);
 output_stats('CCW',[CCW_hup_T1; CCW_dump_T1],[CCW_hup_T2;
CCW_dump_T2]);
 output_stats('head-up',[CW_hup_T1; CCW_hup_T1],[CW_hup_T2; CCW_hup_T2]);
 output_stats('dump',[CW_dump_T1; CCW_dump_T1],[CW_dump_T2;
CCW_dump_T2]);


 fprintf('\nAverage data across stimulus types:\n');

 for j=1:max(size(session))

  fprintf(['\nData for BDC',int2str(session(j)),':\n']);

  CW_hup_T1 = [];
  CW_hup_T2 = [];
  CCW_hup_T1 = [];
  CCW_hup_T2 = [];
  CW_dump_T1 = [];
  CW_dump_T2 = [];
  CCW_dump_T1 = [];
  CCW_dump_T2 = [];
  for k=1:max(size(runs))
   run_code = [subject(i), int2str(session(j)), runs(k,:)];
   index = find_entry(hash_list,run_code);
   if (index > 0)
    T1 = all_parms(index,4);
    if (T1 ~= NaN)
     if (CW_flag(k))
      if (dump_flag(k))
```

```
          CW_dump_T1 = [CW_dump_T1; T1];
         else
          CW_hup_T1 = [CW_hup_T1; T1];
         end
        else
         if (dump_flag(k))
          CCW_dump_T1 = [CCW_dump_T1; T1];
         else
          CCW_hup_T1 = [CCW_hup_T1; T1];
         end
        end
       end
       T2 = all_parms(index,5);
       if (T2 ~= NaN)
        if (CW_flag(k))
         if (dump_flag(k))
          CW_dump_T2 = [CW_dump_T2; T2];
         else
          CW_hup_T2 = [CW_hup_T2; T2];
         end
        else
         if (dump_flag(k))
          CCW_dump_T2 = [CCW_dump_T2; T2];
         else
          CCW_hup_T2 = [CCW_hup_T2; T2];
         end
        end
       end
      end
    end

    output_stats('CW h-up',CW_hup_T1,CW_hup_T2);
    output_stats('CCW h-up',CCW_hup_T1,CCW_hup_T2);
    output_stats('CW dump',CW_dump_T1,CW_dump_T2);
    output_stats('CCW dump',CCW_dump_T1,CCW_dump_T2);

    output_stats('CW',[CW_hup_T1;CW_dump_T1],[CW_hup_T2;CW_dump_T2]);

output_stats('CCW',[CCW_hup_T1;CCW_dump_T1],[CCW_hup_T2;CCW_dump_T2]);
    output_stats('head-up',[CW_hup_T1;CCW_hup_T1],[CW_hup_T2;CCW_hup_T2]);

output_stats('dump',[CW_dump_T1;CCW_dump_T1],[CW_dump_T2;CCW_dump_T2])
;

    T1 = [CW_hup_T1; CCW_hup_T1; CW_dump_T1; CCW_dump_T1];
    T2 = [CW_hup_T2; CCW_hup_T2; CW_dump_T2; CCW_dump_T2];
    output_stats('all',T1,T2);

  end
end
```

```
function [] = output_stats(string,T1,T2)

%output_stats
%
% Function used by "mean_butt_data" to print out the mean,
% variance and number of data points across a group of runs,
% for both per- and post-rotatory button push times.
%
% D. Balkwill 9/26/91

fprintf(['\n',string,':\n']);
fprintf('  Mean T1 = %5.2f.\n',mean(T1));
fprintf('  Variance in T1 = %5.2f.\n',std(T1)*std(T1));
fprintf(['    ',int2str(length(T1)),' data points for T1.\n']);
fprintf('  Mean T2 = %5.2f.\n',mean(T2));
fprintf('  Variance in T2 = %5.2f.\n',std(T2)*std(T2));
fprintf(['    ',int2str(length(T2)),' data points for T2.\n']);

return;
```

Appendix H

Statistical Preparation Software

stat_prep

log_outlier

mag_outlier

dec30_spv

lin_fit

pack_true

search

```
%stat_prep
%
% Prepares an SPV profile for statistical analysis. The first
% step is time-shifting and stripping out extra data to leave
% one minute per-rotatory and one minute post-rotatory. The
% second step is outlier detection. The third step is decimation
% by a factor of 30 down to 4 Hz.
%
% D. Balkwill 8/8/91

run_code = input('Enter Run Code: ','s');

%get sampling frequency
nysa_path = get_path;
eval(['load ',nysa_path,'bookkeeping:vel_filter.mat']);
clear A B
minute_size = 60 * sample;

%load data
file_specs
data_path = [data_path,run_code(1),':'];
eval(['load ',data_path,file_name(Parms_File,run_code)]);
eval(['load ',data_path,file_name(AVel1_File,run_code)]);
eval(['x = ',AVel1_Var,';']);
eval(['clear ',AVel1_Var]);

%
% Normalize SPV profile to one-minute per-rotatory and
% one-minute post-rotatory phase.
%

y = zeros(1,2 * minute_size + 1);   %initialize to two minutes

delay = delay * sample;
spinl = spinl * sample;

if (spinl >= minute_size)  %extract first minute of per-rotatory
   y(1:minute_size) = x(delay:(delay+minute_size-1));
else    %pad per-rotatory out to one minute
   y(1:spinl) = x(delay:(delay+spinl-1));
   y(spinl+1:minute_size) = ones(1,minute_size-spinl) * median(x((delay+spinl-
5):(delay+spinl-1)));
end

%post-rotatory data
y((minute_size+1):(2*minute_size+1)) = x((delay+spinl):(delay+spinl+minute_size));

delay = delay / sample;
spinl = spinl / sample;

clear x

%
% Determine sections to be excluded from statistical analysis
```

```
% (dropouts and outliers).
%

t = [1:(2*minute_size+1)] / sample;
good_data = ones(1,2*minute_size+1);

%
%find valid range for log outlier detection in per-rotatory section
%
fprintf('Per-rotatory:\n');
i = sample + 1;  %one second after start
j = i + 20 * sample;    %insist upon 20 seconds, minimum
while (abs(mean(y(j:j+5*sample))) > 10)  %look for mean spv under 10 deg/sec
   j = j + 2 * sample;
end
t1 = t(i:j);
y1 = y(i:j);

[logout1,under,over,m,b] = log_outlier(t1,y1,run_code);

%take care of under- or over-flow
good_data(i:j) = ~logout1;
if (under > 0)
   good_data(i-under:i-1) = zeros(1,under);
end
if (over > 0)
   good_data(j+1:j+over) = zeros(1,over);
end

%save final fit in parms file as first-order "model fit"
tau1 = -1/m;
gain1 = exp(b)/120;
fprintf('Time length of outliers from log fit is ');
fprintf('%5.2f seconds.\n',(sum(logout1)+under+over)/sample);

% do magnitude outlier detection on remainder of per-rotatory SPV

i = j + over + 1;
[magout1,under,over] = mag_outlier(t(i:minute_size),y(i:minute_size),30);
good_data(i:minute_size) = ~magout1;
if (under > 0)
   good_data(i-under:i-1) = zeros(1,under);
end
fprintf('Time length of outliers from magnitude threshold is');
fprintf(' %5.2f seconds.\n',(sum(magout1)+under)/sample);
% don't fill in any overflow, because this would be post-rotatory


%
%find valid range for outlier detection in post-rotatory section
%
fprintf('Post-rotatory:\n');
i = minute_size + sample + 1;  %one second after stop
j = i + 20 * sample;    %insist upon 20 seconds, minimum
```

```
while (abs(mean(y(j:j+5*sample))) > 10)
   j = j + 2 * sample;
end
t2 = t(i:j);
y2 = y(i:j);

[logout2,under,over,m,b] = log_outlier(t2,y2,run_code);

%take care of under- or over-flow
good_data(i:j) = ~logout2;
if (under > 0)
   good_data(i-under:i-1) = zeros(1,under);
end
if (over > 0)
   good_data(j+1:j+over) = zeros(1,over);
end

%save final fit in parms file as first-order "model fit"
tau2 = -1/m;
gain2 = exp(b + 60 * m)/120;
fprintf('Time length of outliers from log fit is ');
fprintf('%5.2f seconds.\n',(sum(logout2)+under+over)/sample);

% do magnitude outlier detection on remainder of per-rotatory SPV

i = j + over + 1;
[magout2,under,over] = mag_outlier(t(i:2*minute_size),y(i:2*minute_size),30);
good_data(i:2*minute_size) = ~magout2;
if (under > 0)
   good_data(i-under:i-1) = zeros(1,under);
end
fprintf('Time length of outliers from magnitude threshold');
fprintf(' is %5.2f seconds.\n',(sum(magout2)+under)/sample);
% don't fill in any overflow, because this would be past two minutes

fprintf('Overall percentage of good data is %6.2f\n',100*mean(good_data));


%plot data and outlying regions
hold off
plot(t1,log(abs(y1)))
hold on
plot(t1,log(abs(mean(y1))) * logout1,'b--')
grid
xlabel('Time (sec)')
ylabel('ln(SPV)')
title([run_code,' -- SPV and log outlier indicator'])
pause

hold off
plot(t2,log(abs(y2)))
hold on
plot(t2,log(abs(mean(y2))) * logout2,'b--')
grid
```

```
xlabel('Time (sec)')
ylabel('ln(SPV)')
title([run_code,' -- SPV and log outlier indicator'])
pause

hold off
plot(t,y)
hold on
plot(t,50*(~good_data),'b--')
plot(t,-50*(~good_data),'b--')
grid
xlabel('Time (sec)')
ylabel('SPV (deg/sec)')
title([run_code,' -- SPV and overall outlier indicator'])
pause

hold off
plot(t(good_data),y(good_data),'.')
grid
xlabel('Time (sec)')
ylabel('SPV (deg/sec)')
title([run_code,' -- good SPV (outliers removed)'])
pause

hold off

%save parms files, updated to include "model fit" parameters
eval(['save ',data_path,file_name(Parms_File,run_code),' delay spinl spinv T1 T2 runlen
tau1 gain1 tau2 gain2']);

clear t t1 t2 y1 y2 i j logout1 logout2 minute_size sample
clear delay spinl spinv T1 T2 runlen under over magout1 magout2
clear m b tau1 gain1 tau2 gain2

norm_spv = y;
clear y

% save normalized data, having departed from 'file_specs' at this point
eval(['save ',data_path,run_code,'.good good_data']);
eval(['save ',data_path,run_code,'.norm norm_spv']);

%decimate data to 4 Hz, and save decimated information
dec30_spv

clear_specs
clear good_data norm_spv data_path nysa_path run_code
```

```
function [diff,underflow,overflow,m,b] = log_outlier(t,x,run_code)

% Function to perform outlier detection based upon the natural
% logarithm of the SPV data. The data is log-transformed, an
% optimal linear fit is calculated, and the RMS of the data
% about this line calculated. Any difference exceeding 6 RMS
% values is flagged as bad data (outlying artifact). Any
% difference exceeding 3 RMS values, at a point where the
% data point is less than 2 log units (7.4 deg/s^2) is flagged
% as bad data (dropout). This only works on data which is
% greater than 8 deg/sec^2. Any point within half a second of
% a bad data point is marked as bad, to account for transient
% behaviour.
%
% A new linear fit is calculated for the good data, new RMS
% calculated, and bad data flagging is repeated. This process
% reiterates until RMS converges to within 20% on subsequent
% iterations.
%
% The log-transformed data is displayed, along with each linear
% fit, and the 3 RMS error bar lines.
%
% D. Balkwill 8/8/91

axis([t(1) t(length(t)) -100 50]);
plot(t,x)
xlabel('Time (sec)')
ylabel('SPV (deg/sec)')
title([run_code,' -- Outlier Detection Algorithm'])
pause

sample = round( 1 / (t(2) - t(1)) );   %get sample rate
filt_length = 1 + (sample/2);

[m1,m2] = size(x);
if (m1 > 1)
   x = x';
end
l = max(m1,m2);    %get number of samples

s = sign(mean(x));   % predominant direction of SPV
bool = ((s * x) > 0);   % 1 iff SPV is in predominant direction
filler = 1e-10 * ones(1,l);
whos

% x is log(SPV) if x in predominant direction, or 1e-10 otherwise
x = log( (s * bool .* x) + ((~bool) .* filler) );

%do a first pass on the outlier detection
pass = 1;
[m,b] = lin_fit(t,x);
y = m * t + b;
d = abs(y - x);
RMS = sqrt(mean(d .* d));
```

```
fprintf('Pass %2.0f:  slope = %7.4f,  ',pass,m);
fprintf('intercept = %6.3f,  RMS = %6.3f\n',b,RMS);

hold off
clg
axis([t(1) t(l) -5 5]);
plot(t,x,':')
hold on
plot(t,y,'-')
plot(t,y+(3*RMS),'--')
plot(t,y-(3*RMS),'--')
xlabel('Time (sec)')
ylabel('Ln (SPV)')
title([run_code,' -- Outlier Detection Algorithm'])
%pause

bool = ((d > min(10,(3 * RMS))) & (x < 2)) | (d > (6 * RMS));
diff = filtfilt(ones(1,filt_length),1,bool);
diff = (diff > 0);   % diff is 1 iff there is an outlier

oldRMS = 1e50;   %dummy assignment
while (abs((oldRMS - RMS)/RMS) > 0.2)

   oldRMS = RMS;
   pass = pass + 1;

   good = pack_true(~diff);

   % next curve fit on good data only

   [m,b] = lin_fit(t(good),x(good));

   clear y
   y = m * t + b;
   RMS = sqrt(mean(d(good) .* d(good)));
    fprintf('Pass %2.0f:  slope = %7.4f,  ',pass,m);
    fprintf('intercept = %6.3f,  RMS = %6.3f\n',b,RMS);

    plot(t,y,'-')
    plot(t,y+(3*RMS),'--')
    plot(t,y-(3*RMS),'--')
%pause

   % remark bad data points based on new values
   % Note: all points, even previously good ones, are checked
   clear d bool diff
   d = abs(y - x);
   bool = ((d > min(10,(3 * RMS))) & (x < 2)) | (d > (6 * RMS));
   diff = filtfilt(ones(1,filt_length),1,bool);
   diff = (diff > 0);   % diff is 1 iff there is an outlier

end
hold off
```

```
% look for outlier in previous half second of data
for i = 1:filt_length
   if (bool(i) > 0)
      break;
   end
end
% number of extra outlying data points before this section of data
underflow = filt_length - i;

% look for outlier in last half second of data
for i = 1:filt_length
   if (bool(l-i+1) > 0)
      break;
   end
end
% number of extra outlying data points after this section of data
overflow = filt_length - i;

return;
```

```
function [diff,underflow,overflow] = mag_outlier(t,x,thresh)

% Function to perform outlier detection based upon a specified
% magnitude threshold.  Any data point exceeding this threshold
% is marked as bad data (artifact).  Any point within half a
% second of a bad data point is marked as bad, to account for
% transient behaviour.
%
% D. Balkwill 8/8/91

sample = round( 1 / (t(2) - t(1)) );
filt_length = 1 + (sample/2);

[m1,m2] = size(x);
if (m1 > 1)
   x = x';
end
l = max(m1,m2);

x = abs(x);
bool = (x > thresh);
diff = filtfilt(ones(1,filt_length),1,bool);
diff = (diff > 0);

for i = 1:filt_length
   if (bool(i) > 0)
      break;
   end
end
underflow = filt_length - i;

for i = 1:filt_length
   if (bool(l-i+1) > 0)
      break;
   end
end
overflow = filt_length - i;

return;
```

```
%dec30_spv
%
% Decimates SPV data by a factor of 30, from 120 to 4 Hz.
% The output of this routine is a time series in which each
% SPV point is the mean of 30 samples (one-quarter second).
% "Bad data" is taken into account by averaging only across
% the good data, so that some points may be an average of
% fewer than 30 points.  If an entire quarter-second is bad,
% the decimated SPV "dec_spv", variance "within", and "good_data"
% flag will all be zero.  Notice that no formal low-pass
% filtering is done on the data.  Also, each decimated sample
% is independent of the others since the boxcar averager is
% such that none of the original data points are included in
% the calculation of more than one decimated sample.
%
% D. Balkwill 8/8/91

% set bad data to zero for summation purposes
norm_spv = norm_spv .* good_data;

l = length(good_data);
new_l = (l - 1)/30;  %new sampling frequency
x = zeros(30,new_l);  %matrix form of SPV data
n = zeros(1,new_l);

for i=1:30
   x(i,:) = norm_spv(i:30:(l+i-30));  %column is every 30th sample
   n = n + good_data(i:30:(l+i-30));  %whether data is good or not
end

dec_good = (n > 0);   %good data flag

% Decimated SPV is box-car average across row, with n=number of
% good samples.  Correction in denominator to prevent division
% by zero for an entire bin of bad data;  dec_spv=0 in this case.
dec_spv = sum(x) ./ (n + (~dec_good));

% Variance within trace is variance of each bin, where denominator
% contains correction to prevent division by zero when number of
% good samples in bin is <= 1;  variance within=0 in this case.
within = sum(x .* x) - (n .* dec_spv .* dec_spv);
within = abs(within) ./ (n - 1 + (2 * (n <= 1)));

%save data, having departed from 'file_specs' by now
eval(['save ',data_path,run_code,'.dec_spv dec_spv']);
eval(['save ',data_path,run_code,'.dec_good dec_good']);
eval(['save ',data_path,run_code,'.within within']);

clear dec_good dec_spv i l n new_l within x
```

```
%lin_fit
%
% D. Balkwill 5/21/91
% This determines a linear curve fit, y = mt + b, to a segment,
% x, by calculating optimal m and b. The error function to
% minimize is the sum of squares:
%     sum{ [(mt + b) - x]^2 }
% t is the sequence of time values
% x is the sequence of actual values at each time
% y is the sequence of curve fit points
%
% To minimize with respect to m and b, we require the partial
% derivatives to be zero, yielding:
%     sum{ 2[(mt + b) - x]t } = 0       d/dm
%     sum{ 2[(mt + b) - x] }  = 0       d/db
%
% Hence,
%       m * sum{ t^2 } + b * sum{ t } = sum{ xt }
%       m * sum{ t }   + b * sum{ 1 } = sum{ x }
%
% Therefore,
%       m = [ sum{ xt } * sum{ 1 } - sum{ t } * sum{ x }]
%         / [ sum{ t^2 } * sum{ 1 } - sum{ t } * sum{ t }]
%       b = [ sum{ t^2 } * sum{ x } - sum{ xt } * sum{ t }]
%         / [ sum{ t^2 } * sum{ 1 } - sum{ t } * sum{ t }]

function [m,b] = lin_fit(t,x)

[n1,n2] = size(t);
[n3,n4] = size(x);

if (n1 ~= n3)
   t = t';
   [n1,n2] = size(t);
end
l1 = n1 * n2;
l2 = n3 * n4;
if (l1 >= l2)
   l = l2;       % sum{ 1 }
   t = t(1:l);
else
   l = l1;       % sum{ 1 }
   x = x(1:l);
end

A = sum(t .* t);   % sum{ t^2 }
B = sum(t);        % sum{ t }
C = sum(x);        % sum{ x }
D = sum(x .* t);   % sum{ xt }

m = (D * l - B * C) / (A * l - B * B);
b = (A * C - D * B) / (A * l - B * B);

return;
```

```
function indices = pack_true(bool)

% returns vector of indices corresponding to samples which are
% true (1), given a boolean time series, bool.
%
% D. Balkwill 8/8/91

l = length(bool);

i = search(bool,0,1);   %find start of first false section
if (i == 1)
   indices = [];
elseif (i == NaN)
   indices = [1:l];
else
   indices = [1:i-1];
end

while (i ~= NaN)
   i = search(bool,1,i);   %look for beginning of true section
   if (i ~= NaN)
      beg = i;
      i = search(bool,0,i);  %look for beginning of false section
      if (i ~= NaN)
         indices = [indices, (beg:i-1)];   %add true section
      else
         indices = [indices, (beg:l)];     %add true section
      end
   end
end

return;
```

```
function index = search(vec,val,start)

% Searches a vector (vec) for a value (val), starting at a given
% index (start), and returns the index at which the value was
% found.  The value NaN is returned if the value was not found.
%
% D. Balkwill 8/8/91

l = length(vec);
cont = 1;
index = start;
while (cont)
   if (vec(index) == val)
      cont = 0;
   elseif (index == l)
      cont = 0;
      index = NaN;
   else
      index = index + 1;
   end
end
return;
```

Appendix I

Statistical Analysis Software

dec_mean

combine_per_post

new_tsq

tsq_pvalues

spv_stats

new_compare_spv

```
%dec_mean
%
% This routine averages together a user-specified number of runs,
% to produce a mean SPV response, and a variance of that response.
% The run codes must be entered, along with a code for the "stats"
% file which contains the mean, variance, and number of data
% samples at each time point.
%
% Written by B.McGrath
% Revised 4/91 by C. Pouliot
% Rewritten 7/91 by D. Balkwill
% Revised 11/6/91 by D. Balkwill

num_runs = input('Number of Runs >> ');

for i = 1:num_runs
        eval(['run_code',int2str(i),' = input("Enter Run Code: ","s");'])
end

stat_code = input('Enter code for stats file: ','s');

nysa_path = get_path;

sample = 4;
minute_size = 60 * sample;

sum_spv = zeros(1,2*minute_size);
sum_square = zeros(1,2*minute_size);
total = zeros(1,2*minute_size);

file_specs
eval(['chdir ',data_path]);

for i = 1:num_runs
   run_code = eval(['run_code',int2str(i)]);
   eval(['clear run_code',int2str(i)]);
   if (exist([run_code,'.dec_spv']) ~= 2)
      fprintf(['Error: cannot find data for run code ',run_code]);
   else
       eval(['load ',run_code,'.dec_spv']);
       eval(['load ',run_code,'.dec_good']);
      sum_spv = sum_spv + (dec_spv .* dec_good);
      sum_square = sum_square + (dec_spv .* dec_spv .* dec_good);
      total = total + dec_good;
      clear dec_spv dec_good
   end
end

mean_spv = sum_spv ./ (total + (total <= 0));  %ensure mean <> NaN
var_spv = (sum_square - (sum_spv .* mean_spv));
var_spv = var_spv ./ ((total - 1) + 2*(total <= 1));  %ensure var <> NaN
clear sum_spv sum_square

eval(['save ',stat_code,'.stats mean_spv var_spv total']);
```

```
hold off
clg
std_spv = sqrt(var_spv);
t = ([1:length(mean_spv)] - 0.5) / sample;

axis([0 120 min(mean_spv-std_spv) max(mean_spv+std_spv)]);
plot(t,mean_spv);
hold on
plot(t,mean_spv-std_spv,'g')
plot(t,mean_spv+std_spv,'g')
grid
title([stat_code,' -- Mean SPV +- standard deviation']);
xlabel('Time (sec)');
ylabel('SPV (deg/sec)');
hold off

%clear total mean_spv var_spv std_spv
clear stat_code run_code i num_runs sample minute_size
clear_specs
clear nysa_path data_path
```

```
%combine_per_post
%
% Combines average per-rotatory and post-rotatory SPV data into
% one profile.  This allows different runs to be included in
% the per-rotatory and post-rotatory portions of the profile.
%
% D. Balkwill  11/6/91

stat_code = input('Enter code for stats file: ','s');

file_specs
eval(['chdir ',data_path]);

load per.stats
per_mean = mean_spv;
per_total = total;
per_var = var_spv;

load post.stats
mean_spv(1:240) = per_mean(1:240);
var_spv(1:240) = per_var(1:240);
total(1:240) = per_total(1:240);

eval(['save ',stat_code,'.stats mean_spv var_spv total']);

clear_specs
clear data_path per_total per_mean per_var
```

```
%new_tsq
%
% This routine performs a Monte Carlo simulation to determine
% the sum-of-t-squares distribution. The simulation is done for
% a grid of values for N1 ranging from 2 to 20 in increments of
% 2, and integer N2 ranging from 1 to 10. The number of degrees
% of freedom is 100, and it is repeated for 5000 iterations.
% Note: The time required on a Mac II for this simulation is
% approximately 40 seconds per iteration.
%
% C. Pouliot and D. Balkwill 12/6/91

N1 = 10;
N2 = 10;

n1 = 2 * [1:N1]';
n2 = [1:N2];
Mn1 = n1 * ones(1,N2);
Mn2 = ones(N1,1) * n2;

for i=1:N1
   for j=1:N2
      eval(['dist',int2str(n1(i)),'_',int2str(n2(j)),' = zeros(1000,1);']);
   end
end
rand('normal');

eps1 = zeros(N1,1);
eps1sq = zeros(N1,1);
ssq1 = zeros(N1,1);
eps2 = zeros(1,N2);
eps2sq = zeros(1,N2);
ssq2 = zeros(1,N2);

for total = 1:5000

   time = fix(clock);
   fprintf('%2.0f:',time(4));
   fprintf('%2.0f:',time(5));
   fprintf('%2.0f   ',time(6));
   fprintf('total = %5.0f\n',total);

   tsq_sum = zeros(N1,N2);

   for number = 1:100

      r1 = rand(2 * N1);
      for i=1:N1
         rand1 = r1(2*i,(1:(2*i)));
         rand1sq = rand1 .* rand1;
         eps1(i) = sum(rand1);
         eps1sq(i) = sum(rand1sq);
      end
```

```
      r2 = rand(N2);
      for j=1:N2
         rand2 = r2(1:j,j);
         rand2sq = rand2 .* rand2;
         eps2(j) = sum(rand2);
         eps2sq(j) = sum(rand2sq);
      end

      mn1 = eps1 ./ n1;
      mn2 = eps2 ./ n2;
      ssq1 = eps1sq - (eps1 .* mn1);
      ssq2 = eps2sq - (eps2 .* mn2);

      Mmn1 = mn1 * ones(1,N2);
      Mssq1 = ssq1 * ones(1,N2);

      Mmn2 = ones(N1,1) * mn2;
      Mssq2 = ones(N1,1) * ssq2;

      vardif = ((Mssq1 + Mssq2) ./ (Mn1 + Mn2 - 2)) .*
((ones(N1,N2)./Mn1)+(ones(N1,N2)./Mn2));
      tsq = (((Mmn1) - (Mmn2)) .^ 2) ./ vardif;

      tsq_sum = tsq_sum + tsq;
   end

   val = round(tsq_sum);
   bool = (val > 1000);
   val = (val .* (~bool)) + (bool * 1000);

   for i=1:N1
      for j=1:N2
         eval(['dist',int2str(n1(i)),'_',int2str(n2(j)),'(val(i,j)) =
dist',int2str(n1(i)),'_',int2str(n2(j)),'(val(i,j)) + 1;']);
      end
   end

end

eval(['save tsq_dist']);
```

```
%tsq_pvalues
%
% This routine loads in the distributions which were simulated
% by "new_tsq", and calculates the precentiles for each (N1,N2)
% combination corresponding to p = 0.5, 0.975, 0.99, 0.995, and
% 0.999.  These percentiles are saved on the disk for future
% use and interpolation by "spv_stats".
%
% D. Balkwill 12/6/91

load Wonko:tsq_dist

L = length(dist8_4);
N = sum(dist8_4);

N1 = 10;
N2 = 10;

n1 = 2 * [1:N1]';
n2 = [1:N2];
Mn1 = n1 * ones(1,N2);
Mn2 = ones(N1,1) * n2;

p50 = L * ones(N1,N2);
p05 = L * ones(N1,N2);
p025 = L * ones(N1,N2);
p01 = L * ones(N1,N2);
p005 = L * ones(N1,N2);
p001 = L * ones(N1,N2);

thresh50 = .50 * N;
thresh05 = .95 * N;
thresh025 = .975 * N;
thresh01 = .99 * N;
thresh005 = .995 * N;
thresh001 = .999 * N;

for i=1:N1
   fprintf('\n%2.0f:',n1(i));
  for j=1:N2
     fprintf(' %2.0f',n2(j));
      eval(['dist = dist',int2str(n1(i)),'_',int2str(n2(j)),';']);
     plot(dist)
     total = 0;
     for k50=1:L
        total = total + dist(k50);
        if (total > thresh50)
           p50(i,j) = k50;
           break;
        end
     end
     for k05=(k50+1):L
        total = total + dist(k05);
        if (total > thresh05)
```

```
            p05(i,j) = k05;
            break;
         end
      end
      for k025=(k05+1):L
         total = total + dist(k025);
         if (total > thresh025)
            p025(i,j) = k025;
            break;
         end
      end
      for k01=(k025+1):L
         total = total + dist(k01);
         if (total > thresh01)
            p01(i,j) = k01;
            break;
         end
      end
      for k005=(k01+1):L
         total = total + dist(k005);
         if (total > thresh005)
            p005(i,j) = k005;
            break;
         end
      end
      for k001=(k005+1):L
         total = total + dist(k001);
         if (total > thresh001)
            p001(i,j) = k001;
            break;
         end
      end
   end
end
```

```
%spv_stats
%
% This routine loads in the pre-flight and post-flight mean SPV
% results for a particular subject, and performs various sum-t-sq
% tests on the profiles.  The portions of the traces to be
% compared exclude the first second after the start or stop of
% the chair (since the chair is still moving) and extend for
% forty seconds after that.
%
% Comparisons within the pre-flight responses (and also within
% the early post-flight responses) look for differences between
% head-up and dumping responses (post-rotatory only) and between
% CW and CCW responses (per-rot, head-up post-rot, and dumping
% post-rot).  It is tacitly assumed that head-up per-rot and
% dumping per-rot responses will be the same.
%
% Modification: 11/25/91
% Now checks the CW per-rot against the CCW post-rot, and the
% CCW per-rot against the CW post-rot to see if there is a
% difference between per-rot and post-rot responses.
%
% Modification: 11/25/91
% Outputs the critical sum-of-t-squares value for the appropriate
% n1 and n2, based upon linear interpolation within the p025
% table.  This is output instead of the raw sum-of-t-squares.
%
% D. Balkwill  11/25/91

% load p025 table for 5% significance
load Wonko:MatLab:p025

% change path to correct data folder

subject = input('Enter Subject Letter: ','s');
file_specs
clear_specs
l = length(data_path);
if (data_path(l-2) == ':')
   eval(['chdir ',data_path(1:l-2),subject,':']);
else
   eval(['chdir ',data_path,subject,':']);
end


% set time base to 120 seconds at 4 Hz

t = ([1:480] - 0.5)/4;

%
% load all data from disk
%

% pre-flight
eval(['load ',subject,'preCWhup.stats']);
```

```
preCW_spv = mean_spv;
preCW_var = var_spv;
preCW_total = total;
eval(['load ',subject,'preCCWhup.stats']);
preCCW_spv = mean_spv;
preCCW_var = var_spv;
preCCW_total = total;
eval(['load ',subject,'preCWdump.stats']);
preCWd_spv = mean_spv;
preCWd_var = var_spv;
preCWd_total = total;
eval(['load ',subject,'preCCWdump.stats']);
preCCWd_spv = mean_spv;
preCCWd_var = var_spv;
preCCWd_total = total;

% R+0/1/2 (return)
if (subject == 'N')
   session = '78';
else
   session = '67';
end
eval(['load ',subject,session,'CWhup.stats']);
postCW_spv = mean_spv;
postCW_var = var_spv;
postCW_total = total;
eval(['load ',subject,session,'CCWhup.stats']);
postCCW_spv = mean_spv;
postCCW_var = var_spv;
postCCW_total = total;
eval(['load ',subject,session,'CWdump.stats']);
postCWd_spv = mean_spv;
postCWd_var = var_spv;
postCWd_total = total;
eval(['load ',subject,session,'CCWdump.stats']);
postCCWd_spv = mean_spv;
postCCWd_var = var_spv;
postCCWd_total = total;

% R+4/7 (recovery)
eval(['load ',subject,'910CWhup.stats']);
recCW_spv = mean_spv;
recCW_var = var_spv;
recCW_total = total;
eval(['load ',subject,'910CCWhup.stats']);
recCCW_spv = mean_spv;
recCCW_var = var_spv;
recCCW_total = total;
eval(['load ',subject,'910CWdump.stats']);
recCWd_spv = mean_spv;
recCWd_var = var_spv;
recCWd_total = total;
eval(['load ',subject,'910CCWdump.stats']);
recCCWd_spv = mean_spv;
```

```
recCCWd_var = var_spv;
recCCWd_total = total;

%
% Pre-flight baseline tests
%

fprintf('\n  Constant          Pop#1    Pop#2    dof   ratio   crit    N1    N2\n')
fprintf('  ==========        =======  =======   ===  =======  ======
====  ====\n')

%
% pre-flight CW head-up vs. dumping
%
fprintf('Pre-flight CW post    head-up    dump     ')
new_compare_spv( t, preCW_spv, preCW_var, preCW_total, preCWd_spv,
preCWd_var, preCWd_total, 245, 400, p025);

%
% pre-flight CCW head-up vs. dumping
%
fprintf('Pre-flight CCW post   head-up    dump     ')
new_compare_spv( t, preCCW_spv, preCCW_var, preCCW_total, preCCWd_spv,
preCCWd_var, preCCWd_total, 245, 400, p025);

%
% pre-flight CW head-up vs. CCW head-up
%
fprintf('Pre-flight per          CW       CCW      ')
%new_compare_spv( t, preCW_spv, preCW_var, preCW_total, -preCCW_spv,
preCCW_var, preCCW_total, 5, 160, p025);
new_compare_spv( t, recCW_spv, recCW_var, recCW_total, -recCCW_spv,
recCCW_var, recCCW_total, 5, 160, p025);
fprintf('Pre-flight head-up post  CW       CCW      ')
%new_compare_spv( t, preCW_spv, preCW_var, preCW_total, -preCCW_spv,
preCCW_var, preCCW_total, 245, 400, p025);
new_compare_spv( t, recCW_spv, recCW_var, recCW_total, -recCCW_spv,
recCCW_var, recCCW_total, 245, 400, p025);

%
% pre-flight CW dump vs. CCW dump
%
fprintf('Pre-flight dump post    CW       CCW      ')
%new_compare_spv( t, preCWd_spv, preCWd_var, preCWd_total, -preCCWd_spv,
preCCWd_var, preCCWd_total, 245, 400, p025);
new_compare_spv( t, recCWd_spv, recCWd_var, recCWd_total, -recCCWd_spv,
recCCWd_var, recCCWd_total, 245, 400, p025);

%
% pre-flight per-rot vs. post-rot
%
fprintf('Pre-flight          CW per    CCW post    ')
new_compare_spv( t, preCW_spv, preCW_var, preCW_total, preCCW_spv(241:480),
preCCW_var(241:480), preCCW_total(241:480), 5, 160, p025);
```

```
fprintf('Pre-flight          CCW per   CW post    ')
new_compare_spv( t, preCCW_spv, preCCW_var, preCCW_total, preCW_spv(241:480),
preCW_var(241:480), preCW_total(241:480), 5, 160, p025);


%
% Within return sessions tests
%
fprintf('------------------------------------------------------------------------\n')

%
% return CW head-up vs. dumping
%
fprintf('Return CW post       head-up    dump     ')
new_compare_spv( t, postCW_spv, postCW_var, postCW_total, postCWd_spv,
postCWd_var, postCWd_total, 245, 400, p025);

%
% return CCW head-up vs. dumping
%
fprintf('Return CCW post      head-up    dump     ')
new_compare_spv( t, postCCW_spv, postCCW_var, postCCW_total, postCCWd_spv,
postCCWd_var, postCCWd_total, 245, 400, p025);

%
% return CW head-up vs. CCW head-up
%
fprintf('Return per           CW       CCW      ')
new_compare_spv( t, postCW_spv, postCW_var, postCW_total, -postCCW_spv,
postCCW_var, postCCW_total, 5, 160, p025);
fprintf('Return head-up post  CW       CCW      ')
new_compare_spv( t, postCW_spv, postCW_var, postCW_total, -postCCW_spv,
postCCW_var, postCCW_total, 245, 400, p025);

%
% pre-flight CW dump vs. CCW dump
%
fprintf('Return dump post     CW       CCW      ')
new_compare_spv( t, postCWd_spv, postCWd_var, postCWd_total, -postCCWd_spv,
postCCWd_var, postCCWd_total, 245, 400, p025);

%
% pre-flight per-rot vs. post-rot
%
fprintf('Return               CW per   CCW post   ')
new_compare_spv( t, postCW_spv, postCW_var, postCW_total,
postCCW_spv(241:480), postCCW_var(241:480), postCCW_total(241:480), 5, 160,
p025);
fprintf('Return               CCW per  CW post    ')
new_compare_spv( t, postCCW_spv, postCCW_var, postCCW_total,
postCW_spv(241:480), postCW_var(241:480), postCW_total(241:480), 5, 160, p025);


%
```

```
% pre-flight vs. return changes
%
fprintf('------------------------------------------------------------------------\n')

%
% pre-flight CW head-up vs. return CW head-up
%
fprintf('CW per                pre     return    ')
new_compare_spv( t, preCW_spv, preCW_var, preCW_total, postCW_spv, postCW_var,
postCW_total, 5, 160, p025);
fprintf('CW head-up post       pre     return    ')
new_compare_spv( t, preCW_spv, preCW_var, preCW_total, postCW_spv, postCW_var,
postCW_total, 245, 400, p025);

%
% pre-flight CW dump vs. return CW dump
%
%fprintf('CW dump per           pre     return    ')
%new_compare_spv( t, preCWd_spv, preCWd_var, preCWd_total, postCWd_spv,
%postCWd_var, postCWd_total, 5, 160, p025);
fprintf('CW dump post          pre     return    ')
new_compare_spv( t, preCWd_spv, preCWd_var, preCWd_total, postCWd_spv,
postCWd_var, postCWd_total, 245, 400, p025);

%
% pre-flight CCW head-up vs. return CCW head-up
%
fprintf('CCW per               pre     return    ')
new_compare_spv( t, preCCW_spv, preCCW_var, preCCW_total, postCCW_spv,
postCCW_var, postCCW_total, 5, 160, p025);
fprintf('CCW head-up post      pre     return    ')
new_compare_spv( t, preCCW_spv, preCCW_var, preCCW_total, postCCW_spv,
postCCW_var, postCCW_total, 245, 400, p025);

%
% pre-flight CCW dump vs. return CCW dump
%
%fprintf('CCW dump per          pre     return    ')
%new_compare_spv( t, preCCWd_spv, preCCWd_var, preCCWd_total, postCCWd_spv,
%postCCWd_var, postCCWd_total, 5, 160, p025);
fprintf('CCW dump post         pre     return    ')
new_compare_spv( t, preCCWd_spv, preCCWd_var, preCCWd_total, postCCWd_spv,
postCCWd_var, postCCWd_total, 245, 400, p025);


%
% pre-flight vs. recovery changes
%
fprintf('------------------------------------------------------------------------\n')

%
% pre-flight CW head-up vs. recovery CW head-up
%
fprintf('CW per                pre     recovery  ')
```

```
new_compare_spv( t, preCW_spv, preCW_var, preCW_total, recCW_spv, recCW_var,
recCW_total, 5, 160, p025);
fprintf('CW head-up post        pre    recovery   ')
new_compare_spv( t, preCW_spv, preCW_var, preCW_total, recCW_spv, recCW_var,
recCW_total, 245, 400, p025);

%
% pre-flight CW dump vs. recovery CW dump
%
%fprintf('CW dump per            pre    recovery   ')
%new_compare_spv( t, preCWd_spv, preCWd_var, preCWd_total, recCWd_spv,
%recCWd_var, recCWd_total, 5, 160, p025);
fprintf('CW dump post           pre    recovery   ')
new_compare_spv( t, preCWd_spv, preCWd_var, preCWd_total, recCWd_spv,
recCWd_var, recCWd_total, 245, 400, p025);

%
% pre-flight CCW head-up vs. recovery CCW head-up
%
fprintf('CCW per                pre    recovery   ')
new_compare_spv( t, preCCW_spv, preCCW_var, preCCW_total, recCCW_spv,
recCCW_var, recCCW_total, 5, 160, p025);
fprintf('CCW head-up post       pre    recovery   ')
new_compare_spv( t, preCCW_spv, preCCW_var, preCCW_total, recCCW_spv,
recCCW_var, recCCW_total, 245, 400, p025);

%
% pre-flight CCW dump vs. recovery CCW dump
%
%fprintf('CCW dump per           pre    recovery   ')
%new_compare_spv( t, preCCWd_spv, preCCWd_var, preCCWd_total, recCCWd_spv,
%recCCWd_var, recCCWd_total, 5, 160, p025);
fprintf('CCW dump post          pre    recovery   ')
new_compare_spv( t, preCCWd_spv, preCCWd_var, preCCWd_total, recCCWd_spv,
recCCWd_var, recCCWd_total, 245, 400, p025);
```

```
function [] = new_compare_spv(t,spv1,var1,tot1,spv2,var2,tot2,n1,n2,plevels)

%new_compare_spv
%
% This function compares two SPV profiles (spv1 and spv2) between
% sample numbers n1 and n2. The variance and total number of
% data points at each point in time are given. These are the
% contents of the ".stats" files created by "stat_prep". The
% time vector is also given. Only those time points at which
% data exists for both traces, and a variance estimate is known,
% are analyzed.
%
% The variances are smoothed by a five-point moving averager, so
% that unusually low variances are removed, and a pooled variance
% estimate is calculated at each point. The number of standard
% deviations between the two traces at each point is calculated,
% and those points at which the t-test yields 5% significance are
% highlighted. The sum of t-square parameter is calculated and
% displayed, along with average 'n', number of degrees of
% freedom, and the ratio of sum-t-square to dof.
%
% Modification: see "spv_stats"
%
% D. Balkwill 11/25/91

t = t(n1:n2);
spv1 = spv1(n1:n2);
var1 = var1(n1:n2);
tot1 = tot1(n1:n2);
spv2 = spv2(n1:n2);
var2 = var2(n1:n2);
tot2 = tot2(n1:n2);
good1 = pack_true(tot1 > 0);
good2 = pack_true(tot2 > 0);

t975 = [12.706, 4.303, 3.182, 2.776, 2.571, 2.447, 2.365, 2.306, 2.262, 2.228, 2.201,
2.179, 2.160, 2.145, 2.131, 2.120, 2.110, 2.101, 2.093, 2.086, 2.080, 2.074, 2.069,
2.064, 2.060, 2.056, 2.052, 2.048, 2.045, 2.042];

axis([floor(min(t)) ceil(max(t)) min([spv1 spv2]) max([spv1 spv2])]);
plot(t(good1),spv1(good1));
hold on
plot(t(good2),spv2(good2),'b');
grid
hold off
xlabel('Time (sec)')
ylabel('SPV (deg/sec)')
%pause

both_good = pack_true((tot1 > 0) & (tot2 > 0) & ((tot1 + tot2) > 2));
t_good = t(both_good);
tot1 = tot1(both_good);
var1 = var1(both_good);
spv1 = spv1(both_good);
```

```
tot2 = tot2(both_good);
var2 = var2(both_good);
spv2 = spv2(both_good);

l = length(both_good);
v1 = [var1(1:2) var1 var1(l-1:l)];
var1 = (v1(1:l) + v1(2:l+1) + v1(3:l+2) + v1(4:l+3) + v1(5:l+4)) / 5;
v2 = [var2(1:2) var2 var2(l-1:l)];
var2 = (v2(1:l) + v2(2:l+1) + v2(3:l+2) + v2(4:l+3) + v2(5:l+4)) / 5;

df = (tot1 + tot2 - 2);
var_pooled = (((tot1 - 1) .* var1) + ((tot2 - 1) .* var2)) ./ df;
var_pooled = var_pooled + (var_pooled == 0);  %ensure var <> 0
weight = (tot1 + tot2) ./ (tot1 .* tot2);   %1/tot1 + 1/tot2
num_std = ((spv1 - spv2) ./ sqrt(weight .* var_pooled));

plot(t_good,num_std);
hold on
plot(t_good,t975(df),'g')
plot(t_good,-t975(df),'g')
hold off
xlabel('Time (sec)')
ylabel('Number of standard deviations difference')
grid
pause

different = pack_true(abs(num_std) > t975(df));

axis([floor(min(t)) ceil(max(t)) min([spv1 spv2]) max([spv1 spv2])]);
plot(t_good,spv1,'r-')
hold on
plot(t_good,spv2,'b-')
plot(t_good(different),spv1(different),'go');
plot(t_good(different),spv2(different),'wo');
hold off
xlabel('Time (sec)')
ylabel('Different SPV values (deg/sec)')
pause

sumtsq = sum(num_std .* num_std);
n = length(num_std);

mt1 = mean(tot1);
mt2 = mean(tot2);
f1 = floor(mt1);
if (rem(f1,2) == 1)
   f1 = f1 - 1;
end
f2 = floor(mt2);
if1 = f1/2;
if2 = f2;

if (if1 < 1)
   if1 = 1;
```

```
elseif (if1 > 9)
   if1 = 9;
end
if (if2 < 1)
   if2 = 1;
elseif (if2 > 9)
   if2 = 9;
end

x1 = plevels(if1,if2) + ((plevels(if1+1,if2) - plevels(if1,if2)) * (mt1 - f1) / 2);
x2 = plevels(if1,if2+1) + ((plevels(if1+1,if2+1) - plevels(if1,if2+1)) * (mt1-f1) / 2);
crit = x1 + ((x2 - x1) * (mt2 - f2));
crit = crit / 100;

fprintf('%3.0f',n);
fprintf('   %6.2f',sumtsq/n);
fprintf('    %6.2f',crit);
fprintf('   %5.2f',mean(tot1));
fprintf(' %5.2f\n',mean(tot2));
```

Appendix J

Modelling Scripts

mean_model_fit

ind_model_fit

model_err

```
%mean_model_fit
% D. Balkwill 12/9/91
%
% Fits a five-parameter model to SLS-1 FO2 rotating chair SPV data.
% Assumes FO2-dependent tach signal and sampling rate (decimated
% down to 4 Hz). Uses MatLab Optimization Toolbox 'constr'
% function for constrained optimization, with Nelder-Meade
% search algorithm on the parameters. This version fits an
% average SPV profile, fitting the per-rotatory portion first
% and then the post-rotatory portion. The fit is not based upon
% the first two seconds of per- or post-rotatory data.
% The "ind_model_fit" script fits an individual run.

stat_code = input('Enter stats code: ','s');
file_specs
nysa_path = get_path;

%
% load data
%
eval(['chdir ',data_path]);
eval(['load ',stat_code,'.stats']);
dec_good = (total > 0);
save_good = dec_good;
good_indices = pack_true(dec_good);

%
% Initialize time vector, assuming 4 Hz decimated frequency
%
l = length(mean_spv);
t = ([1:l] - 0.5) / 4;
t = t';

%
% shape tach signal with exponential (0.17 sec time constant)
% ramp to a steady state level at 'spinv'
%
Tv = 0.17;
if (rem(l,2) == 1)
   u = [ones(1,(l+1)/2) zeros(1,(l-1)/2)];
else
   u = [ones(1,l/2) zeros(1,l/2)];
end
u = u';

%
% determine spin direction from first 30 seconds of per-rot SPV,
% and set spinv to +- 120 deg/sec for an average profile
%
if (mean(mean_spv(1:120)) < 0)    % CW rotation
   spinv = 120;   % design level
else                              % CCW rotation
   spinv = -120;  % design level
end
```

```
%
% overall control input (tach)
%
u = lsim( spinv/Tv, [1, 1/Tv], u, t);

%
% Nominal model parameters. The parameters to be fitted are the
% non-dimensional ratios of the physical parameters to the
% nominal model parameters here. This places equal emphasis
% on each model parameter, even though they may be orders of
% magnitude apart.
%
K1 = 0.6;
Tc = 6;   %frozen
Ta = 80;  %frozen
h0 = 1 / 30;
g0 = .15;
norm_parms = [K1; Tc; Ta; h0; g0];

options = [1 ; 0.001 ; 0.001];   %error tolerances -- see "help foptions"
vlb = [0.167; 1; 1; 0.1; 0];  %lower bounds
vub = [3; 1; 1; 10; 3]; %upper bounds

plot(t(good_indices),mean_spv(good_indices))

%
% Fit the per-rotatory portion first
%

fprintf(['\n\n\nFitting ',stat_code,' per-rotatory\n']);

dec_good = save_good;
dec_good(1:8) = zeros(1,8);   % do not fit first 2 seconds of data
dec_good(241:480) = zeros(1,240);   % do not fit post-rotatory data
good_indices = pack_true(dec_good);

model_parms = [1; 1; 1; 1; 1];
[model_parms, options] = constr('model_err', model_parms, options, vlb, vub, [], t, u,
mean_spv, good_indices, norm_parms);

model_parms = model_parms .* norm_parms;
eval(['save ',stat_code,'.perfit model_parms options'])

fprintf('*** Model fit: initial model parameters = 1.0\n');
fprintf('Number of iterations = %5.0f\n',options(10));
fprintf('Mean square error = %7.4f\n',options(8));

fprintf('K = %f\n',model_parms(1));
fprintf('Tc = %f\n',model_parms(2));
fprintf('Ta = %f\n',model_parms(3));
fprintf('1/h0 = %f\n',1/model_parms(4));
fprintf('g0 = %f\n',model_parms(5));
```

```
%
% Fit the post-rotatory portion now
%

fprintf(['\n\n\nFitting ',stat_code,' post-rotatory\n']);

dec_good = save_good;
dec_good(1:240) = zeros(1,240);   % do not fit per-rotatory data
dec_good(241:248) = zeros(1,8);   % do not fit first 2 seconds of data
good_indices = pack_true(dec_good);

model_parms = [1; 1; 1; 1; 1];
[model_parms, options] = constr('model_err', model_parms, options, vlb, vub, [], t, u,
mean_spv, good_indices, norm_parms);

model_parms = model_parms .* norm_parms;
eval(['save ',stat_code,'.postfit model_parms options'])

fprintf('*** Model fit: initial model parameters = 1.0\n');
fprintf('Number of iterations = %5.0f\n',options(10));
fprintf('Mean square error = %7.4f\n',options(8));

fprintf('K = %f\n',model_parms(1));
fprintf('Tc = %f\n',model_parms(2));
fprintf('Ta = %f\n',model_parms(3));
fprintf('1/h0 = %f\n',1/model_parms(4));
fprintf('g0 = %f\n',model_parms(5));
```

```
%ind_model_fit
% D. Balkwill 12/9/91
%
% Fits a five-parameter model to SLS-1 FO2 rotating chair SPV data.
% Assumes FO2-dependent tach signal and sampling rate (decimated
% down to 4 Hz). Uses MatLab Optimization Toolbox 'constr'
% function for constrained optimization, with Nelder-Meade
% search algorithm on the parameters. This version fits an
% individual SPV profile, fitting the per-rotatory portion first
% and then the post-rotatory portion. The fit is not based upon
% the first two seconds of per- or post-rotatory data.
% The "mean_model_fit" script fits averaged data.

run_code = input('Enter run code: ','s');
file_specs
nysa_path = get_path;

%
% load data
%
eval(['chdir ',data_path]);
eval(['load ',run_code,'.dec_spv']);
eval(['load ',run_code,'.dec_good']);
eval(['load ',run_code,'.parms']);

save_good = dec_good;
good_indices = pack_true(dec_good);

%
% Initialize time vector, assuming 4 Hz decimated frequency
%
l = length(dec_spv);
t = ([1:l] - 0.5) / 4;
t = t';

%
% shape tach signal with exponential (0.17 sec time constant)
% ramp to a steady state level at 'spinv'
%
Tv = 0.17;
if (rem(l,2) == 1)
   u = [ones(1,(l+1)/2) zeros(1,(l-1)/2)];
else
   u = [ones(1,l/2) zeros(1,l/2)];
end
u = u';

%
% overall control input (tach)
%
u = lsim( spinv/Tv, [1, 1/Tv], u, t);

%
% Nominal model parameters. The parameters to be fitted are the
```

```
% non-dimensional ratios of the physical parameters to the
% nominal model parameters here. This places equal emphasis
% on each model parameter, even though they may be orders of
% magnitude apart.
%
K1 = 0.6;
Tc = 6;   %frozen
Ta = 80;   %frozen
h0 = 1 / 30;
g0 = .15;
norm_parms = [K1; Tc; Ta; h0; g0];

options = [0 ; 0.001 ; 0.001];    %error tolerances -- see "help foptions"
vlb = [0.167; 1; 1; 0.1; 0];  %lower bounds
vub = [3; 1; 1; 10; 3]; %upper bounds

plot(t(good_indices),dec_spv(good_indices))

%
% Fit the per-rotatory portion first
%

fprintf(['\n\n\nFitting ',run_code,' per-rotatory\n']);

dec_good = save_good;
dec_good(1:8) = zeros(1,8);   % do not fit first 2 seconds of data
dec_good(241:480) = zeros(1,240);   % do not fit post-rotatory data

if (sum(dec_good) < 10)
   fprintf('Not enough data points to determine a curve fit.\n');
   return;
end

good_indices = pack_true(dec_good);

model_parms = [1; 1; 1; 1; 1];
[model_parms, options] = constr('model_err', model_parms, options, vlb, vub, [], t, u,
dec_spv, good_indices, norm_parms);

model_parms = model_parms .* norm_parms;
eval(['save ',run_code,'.perfit model_parms options'])

fprintf('*** Model fit: initial model parameters = 1.0\n');
fprintf('Number of iterations = %5.0f\n',options(10));
fprintf('Mean square error = %7.4f\n',options(8));

fprintf('K = %f\n',model_parms(1));
fprintf('Tc = %f\n',model_parms(2));
fprintf('Ta = %f\n',model_parms(3));
fprintf('1/h0 = %f\n',1/model_parms(4));
fprintf('g0 = %f\n',model_parms(5));

%
% Fit the post-rotatory portion now
```

```
%

fprintf(['\n\n\nFitting ',run_code,' post-rotatory\n']);

dec_good = save_good;
dec_good(1:240) = zeros(1,240);   % do not fit per-rotatory data
dec_good(241:248) = zeros(1,8);   % do not fit first 2 seconds of data

if (sum(dec_good) < 10)
   fprintf('Not enough data points to determine a curve fit.\n');
   return;
end

good_indices = pack_true(dec_good);

model_parms = [1; 1; 1; 1; 1];
[model_parms, options] = constr('model_err', model_parms, options, vlb, vub, [], t, u,
dec_spv, good_indices, norm_parms);

model_parms = model_parms .* norm_parms;
eval(['save ',run_code,'.postfit model_parms options'])

fprintf('*** Second fit: initial model parameters = 1.0\n');
fprintf('Number of iterations = %5.0f\n',options(10));
fprintf('Mean square error = %7.4f\n',options(8));

fprintf('K = %f\n',model_parms(1));
fprintf('Tc = %f\n',model_parms(2));
fprintf('Ta = %f\n',model_parms(3));
fprintf('1/h0 = %f\n',1/model_parms(4));
fprintf('g0 = %f\n',model_parms(5));
```

```
function [f,g] = model_err(model_parms,t,u,dec_spv,good_indices,norm_parms)

%model_err
%
% Error function for model fitting.  Constrained optimization
% minimizes the output of this function, which is currently set
% as the mean square error (MSE) between the SPV data and the
% model SPV data.
%
% D. Balkwill 12/9/91

%
% calculate physical parameters for transfer function, and
% determine the corresponding model response
%
model_parms = model_parms .* norm_parms;
K1 = model_parms(1);
Tc = model_parms(2);
Ta = model_parms(3);
h0 = model_parms(4);
g0 = model_parms(5);

num = -[K1, (K1 * (h0 + g0)), 0, 0];
den = [1, (1/Tc + 1/Ta + h0), (1/(Ta*Tc) + h0/Tc + h0/Ta), h0/(Ta*Tc)];
y = lsim(num,den,u,t);

% ensure that y and dec_spv are both either row vectors or column vectors
[m1,n1] = size(y);
[m2,n2] = size(dec_spv);
if (m1 > n1)      % y is column vector
   if (m2 < n2)   % dec_spv is row vector
      y = y';
   end
else              % y is row vector
   if (m2 > n2)   % dec_spv is column vector
      y = y';
   end
end


%
% Only base MSE on data points at which we have valid data.
%
d = y(good_indices) - dec_spv(good_indices);
f = sum(d .* d) / length(d);
%fprintf('MSE = %f\n',f);

plot(t(good_indices),dec_spv(good_indices))
hold on
plot(t(good_indices),y(good_indices),'g');
hold off
```

```
%
% dummy value which 'constr' requires but is unused for our
% purposes;  this must be some constant value for our purposes
%
g = -1;

return;
```